A Reinforcement Learning–Guided General Variable Neighborhood Search for the Traveling Salesman Problem

Karina Assini Andreatta(✉), Maria Claudia Silva Boeres, and Renato Elias Nunes de Moraes

Federal University of Espírito Santo (UFES), Avenue Fernando Ferrari 514, Vitória, ES 29075-910, Brazil
karinassini@gmail.com, {maria.boeres,renato.moraes}@ufes.br
https://www.ufes.br

Abstract. This paper proposes RL-GVNS, a reinforcement learning–guided General Variable Neighborhood Search framework for the Traveling Salesman Problem (TSP), which introduces adaptivity in both the construction and intensification phases. The proposed RL-GVNS extends the classical GVNS by integrating a multi-agent reinforcement learning (MARL) initializer and an RL-driven local search, where neighborhood operators are selected online using Q-learning. Unlike standard GVNS, which relies on a fixed ordering of local search operators, RL-GVNS dynamically adapts its search behavior based on performance feedback, enabling an effective response to non-stationary search dynamics. Although all evaluated configurations, combining classical or MARL-based initialization with standard GVNS or RL-GVNS, perform similarly in small instances, learning-enhanced variants show superior robustness as instance size increases. In particular, the combined MARL + RL-GVNS configuration achieves lower average gaps with respect to the best-known solution, higher best-known solution hit rates, and, in most cases, comparable or reduced execution times relative to non-adaptive GVNS baselines.

Keywords: General Variable Neighborhood Search · Reinforcement Learning · Traveling Salesman Problem

1 Introduction

The Traveling Salesman Problem (TSP) is a classical benchmark in combinatorial optimization. Given the pairwise travel costs between cities, the goal is to find a minimum–cost Hamiltonian tour. Although simply defined, TSP is NP–hard: the search space grows factorially with the number of cities, making exact methods computationally impractical for medium and large–scale instances [23]. This limitation motivates the use of metaheuristic approaches, which yield high-quality solutions at practical computational cost [27]. Among these, Variable

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2027
O. Gervasi et al. (Eds.): ICCSA 2026 Workshops, LNCS 16760, pp. 343–360, 2027.
https://doi.org/10.1007/978-3-032-30521-3_22

Neighborhood Search (VNS), proposed in [26], leverages this trade-off by relying on a systematic exploration of multiple neighborhood structures. The underlying principle of VNS is that a solution that is locally optimal with respect to one neighborhood structure may cease to be optimal when evaluated under a different neighborhood. The method alternates between diversification and intensification by changing the neighborhood during the search process. Diversification is performed through a shaking phase, which applies controlled perturbations to move the search away from the current solution, while intensification is achieved by applying local search procedures within a selected neighborhood to refine the solution and reach a local optimum [21].

In General Variable Neighborhood Search (GVNS), intensification is commonly implemented through Variable Neighborhood Descent (VND) [13]. However, the fixed ordering of local search operators in standard VND leads to static search behavior, motivating adaptive strategies. Although GVNS has been successfully applied to computationally hard optimization problems that arise in practical settings, such as routing, scheduling, and network design [7,18,20,25,34], recent studies have increasingly explored the integration of machine learning techniques to guide metaheuristic decisions, including promising results obtained through reinforcement learning–based operator adaptation [19,29,33].

Despite recent progress, most approaches focus on population-based or evolutionary frameworks [2,17,30], or rely on end-to-end neural construction policies via deep reinforcement learning [22], which require substantial training data and do not readily generalize across instance distributions. Trajectory-based single-solution approaches to the TSP have largely retained fixed operator schemes [7,16,34], with little provision for adapting the selection of local search operators during the run. The underlying difficulty is that TSP instances exhibit non-stationary search landscapes: perturbation moves that reliably yield improvements in early iterations tend to lose their effectiveness as the algorithm approaches high-quality local optima, making a static ordering progressively less efficient as the search converges [21].

As shown in Table 1, the majority of the reviewed approaches incorporate some form of adaptability, although mainly at the population or initialization level. The method proposed here pursues a different direction, placing the adaptation mechanism within the local search phase itself. Q-learning is used to guide the sequencing of neighborhood operators throughout intensification, implementing an Adaptive Operator Selection (AOS) scheme [10] that adjusts operator priorities based on observed performance during the run. The shaking procedure was also modified: at each neighborhood level, a set of operators of increasing structural strength is defined, from which a move is sampled at random, replacing the standard single fixed perturbation and providing a more balanced coverage of the search space across shaking levels.

The remainder of the paper is structured as follows. Section 2 introduces the TSP and reviews the GVNS framework. Section 3 describes the proposed RL-GVNS method, including adaptive initialization, shaking strategy, and local

Table 1. Comparative analysis of related works. Columns indicate: (1) use of VNS or GVNS; (2) population-based approach; (3) explicit parameter calibration or control; (4) adaptive operator selection (AOS); (5) adaptive shaking mechanism; (6) availability of open-source implementation.

Author	(1)	(2)	(3)	(4)	(5)	(6)
da Silva et al. (2010) [7]	✓	✗	–	✗	✗	–
Queiroz dos Santos et al. (2014) [29]	✓	✗	–	✓	✗	–
Todosijevic et al. (2016) [33]	✓	✗	–	✓	✗	–
Menendez et al. (2017) [25]	✓	✗	–	✗	✗	–
Venkatesh et al. (2018) [34]	✓	✗	–	✗	✗	–
Hore et al. (2018) [16]	✓	✗	–	✗	✗	–
Alipour et al. (2018) [2]	✗	✓	–	✗	✗	–
Kool et al. (2019) [22]	✗	✗	–	✗	✗	✓
Karakostas et al. (2019) [20]	✓	✗	Partial	✗	✗	–
Karakostas et al. (2020) [18]	✓	✗	–	✗	✗	–
Karakostas et al. (2022) [19]	✓	✗	–	✓	✓	–
Ruan et al. (2024) [30]	✗	✓	–	✗	✗	–
Jeong et al. (2025) [17]	✗	✓	–	✓	✗	–
This work	✓	✗	✓	✓	✓	✓

search driven by reinforcement learning. Computational experiments and results are presented in Sect. 4. Finally, Sect. 5 summarizes the main findings and discusses directions for future work.

2 Preliminaries

2.1 Traveling Salesman Problem (TSP)

The Traveling Salesman Problem (TSP) is a canonical problem in combinatorial optimization and a standard benchmark for heuristic and metaheuristic algorithms [27]. In this work, the symmetric variant is considered and defined on a complete weighted graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$ represents the set of cities and each edge $(i, j) \in E$ is associated with a non-negative travel cost c_{ij}, with $c_{ij} = c_{ji}$ for all $i, j \in V$.

A feasible solution is a Hamiltonian cycle, represented by a permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ of cities, where π_k denotes the city visited at position k in the tour. In closed tour representations, the starting city may appear duplicated at the end of the sequence. However, the size of the problem n is defined as the number of distinct cities in the tour after removing the duplicated terminal node in the closed representation. The objective is to find a permutation that minimizes the total travel cost along the cycle, including the return to the starting city. Since the size of the search space increases factorially with n, exhaustive enumeration quickly becomes infeasible, motivating the use of heuristic and

metaheuristic approaches that explore neighborhoods defined over permutations to obtain high-quality solutions.

2.2 General Variable Neighborhood Search (GVNS)

GVNS operates over a predefined collection of neighborhood structures and combines a diversification phase, implemented through probabilistic shaking, with a deterministic intensification phase based on VND [13]. In VND, the available neighborhoods are explored in a fixed sequence: upon finding an improving move, the incumbent solution is updated and the search returns to the first neighborhood. If no improvement is found within the current neighborhood, the procedure moves to the next one. This continues until no further improvement is achievable under any available structure, at which point the solution is declared locally optimal. The VND routine is then embedded within an outer loop that alternates between shaking and local search, allowing the algorithm to escape local optima by periodically introducing controlled perturbations of increasing strength before resuming intensification. A complete description of the framework can be found in [12,13].

2.3 Adaptive Operator Selection (AOS)

AOS provides a control layer that regulates the choice of search operators according to their observed contribution during the search [10]. Rather than prescribing a specific learning strategy, it frames operator selection itself as a decision problem, delegating the underlying adaptation mechanism to the method in use. Within the machine learning in metaheuristics (ML-in-MH) paradigm, reinforcement learning has therefore been adopted as a practical means of implementing AOS in search processes that evolve over time [21].

In permutation-based problems such as the TSP, adaptive operator selection has been mainly investigated in population-oriented metaheuristics, including evolutionary algorithms and adaptive large neighborhood search [2,29], while trajectory-based single-solution approaches such as VNS and GVNS have received comparatively limited attention, typically relying on predefined operator schedules throughout the search process.

Reinforcement Learning. Reinforcement learning (RL) [31] is a method in which decisions are adapted over time based on feedback obtained from previous results. Rather than relying on predefined rules, the decision process progressively favors actions that have previously led to improvements, while discouraging those associated with ineffective moves. In the context of neighborhood-based metaheuristics, this allows the selection of local search operators to be guided by their observed contribution to solution quality during the search.

Each local search operator acts as a candidate action, and the quality change it induces in the solution serves as a natural performance signal. Because the usefulness of an operator at a given point in the search depends on the history of

moves already applied, the learning policy must account not only for immediate improvements but also for how operator transitions influence subsequent steps along the trajectory.

This framing places AOS within the broader class of sequential decision problems [10]. After each operator application, the observed outcome is used to update estimates of operator utility, which in turn shape the next selection. Repeated over many steps, this feedback loop gradually shifts effort toward operators that remain productive as the search evolves. Q-learning provides a principled way to close this loop [36], maintaining and updating value estimates for operator transitions without requiring any explicit model of the search landscape.

Q-Learning. Q-learning is an RL method that assigns a value to each possible action according to its expected outcome in a given state. These values are updated iteratively from the observed rewards, allowing the selection of actions that lead to improved results over time without requiring an explicit model of the underlying process.

The learning process is commonly formalized as a Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R)$, where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ the action set, P the transition probabilities, and R the reward function. At each decision step t, the agent observes the current state $s_t \in \mathcal{S}$, selects an action $a_t \in \mathcal{A}$, receives a reward r_t, and transitions to a new state s_{t+1} [35].

Q-learning estimates the action–value function $Q(s, a)$, representing the expected discounted cumulative reward of selecting action a in state s. The estimates are updated iteratively according to the Bellman optimality:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t) \right], \tag{1}$$

where $\alpha \in (0.0, 1.0]$ is the learning rate and $\gamma \in [0.0, 1.0)$ is the discount factor.

Under standard conditions (bounded rewards, Markov property, sufficient exploration), Q-learning converges with probability one to the optimal action–value function [31,35].

3 Proposed Algorithm

The proposed algorithm extends GVNS with RL–driven local search, using continuous credit assignment to adapt operator selection to the evolving search landscape. The Algorithm 1 receives as input an initialization method `InitMethod`, an iteration limit $I_{\max}$, a maximum neighborhood index $k_{\max}$, and a non–improvement limit $P_{\max}$. The initialization method controls how the initial tour is constructed and can assume one of two alternatives: a classical restricted candidate list nearest-neighbor heuristic (RCL) or a learning-based multi-agent reinforcement learning initializer (MARL), both described in detail in Sect. 3.1.

After initializing tour x and computing $c = f(x)$ (lines 1–2), the search proceeds in an outer loop of up to $I_{\max}$ iterations (line 4). Each iteration resets

$k = 1$ (line 7) and explores neighborhoods of increasing size in an inner loop (lines 8–20). At each level k, the incumbent is shaken (line 9) and then intensified by the RL local search (line 11). An improvement ($c'' < c$) triggers an update and resets $k = 1$ to restart from the first neighborhood (lines 12–16), otherwise k is incremented (line 18). The best tour found is returned after all iterations.

Algorithm 1 GVNS with RL-driven Local Search

Require: initialization method `InitMethod`, iteration limit $I_{\max}$, neighborhood limit $k_{\max}$, non-improvement limit $P_{\max}$

1: $x \leftarrow$ INITIALIZE(`InitMethod`)
2: $c \leftarrow f(x)$
3: $p \leftarrow 0$ ▷ consecutive non-improving iterations counter
4: **for** $i = 1$ to $I_{\max}$ **do**
5: **if** $p \geq P_{\max}$ **then break**
6: **end if**
7: $k \leftarrow 1$
8: **while** $k \leq k_{\max}$ **do**
9: $x' \leftarrow$ SHAKE(x, k)
10: $c' \leftarrow f(x')$
11: $(x'', c'') \leftarrow$ RLLOCALSEARCH(x', c')
12: **if** $c'' < c$ **then**
13: $x \leftarrow x''$
14: $c \leftarrow c''$
15: $k \leftarrow 1$ ▷ restart from the first neighborhood
16: $p \leftarrow 0$ ▷ reset non-improvement counter
17: **else**
18: $k \leftarrow k + 1$
19: **end if**
20: **end while**
21: $p \leftarrow p + 1$
22: **end for**
23: **return** x

3.1 Initialization

Two initialization procedures are employed. The first is a nearest–neighbor heuristic with an RCL mechanism [4], which introduces controlled variability in candidate city selection at each construction step. The second is a learning–based strategy inspired by MARL [1], where the tour is assembled incrementally using action–value estimates updated throughout construction.

The learning mechanism is adapted to the single-solution setting of GVNS rather than the population-based framework of [1]. Unlike MARL-based methods that update values only after complete tour evaluations, the proposed initializer employs a Q-learning formulation grounded in the Bellman optimality principle (Eq. 1), incorporating immediate and delayed rewards to account for the influence of early construction choices on subsequent search behavior.

3.2 Adaptive Shaking Strategy

In GVNS, diversification is achieved through a shaking phase that applies random perturbations to the incumbent solution, allowing transitions to neighboring regions of the search space [12]. In the proposed adaptive shaking formulation, (line 9 of Algorithm 1) and presented in Algorithm 2, the neighborhood index k is associated with the intensity of the perturbation by defining, at each shaking level, a set of admissible operators of increasing strength from which a move is randomly sampled.

Algorithm 2 Adaptive Shaking Procedure

Require: Current tour x, shaking strength k
Ensure: Perturbed tour x'
$n \leftarrow |x|$ ▷ number of cities in the tour representation
$\eta_k(n) \leftarrow$ size-dependent parameters ▷ segment lengths, exchange cardinalities
if $k \leq 1$ **then**
  $\mathcal{O}_k \leftarrow$ {Or-opt (small), Shuffle (mild)}
else if $k = 2$ **then**
  $\mathcal{O}_k \leftarrow$ {k-exchange (small), Cross-exchange (mild), Shuffle (medium)}
else if $k = 3$ **then**
  $\mathcal{O}_k \leftarrow$ {Double-bridge, k-exchange (medium), Cross-exchange (medium), Block recombination (medium)}
else
  $\mathcal{O}_k \leftarrow$ {Double-bridge, k-exchange (large), Cross-exchange (strong), Block recombination (strong), Shuffle (strong)}
end if
for a limited number of attempts **do**
  $o \sim \mathcal{U}(\mathcal{O}_k)$
  $x' \leftarrow o(x; \eta_k(n))$
  if $x' \neq x$ **then**
    return x'
  end if
end for
return x

The magnitude of the applied perturbation is controlled through operator parameters expressed as bounded functions of the size of the instance n. Consequently, the shaking phase operates on a set of established TSP perturbation operators, including segment relocations, edge exchanges, non-sequential perturbations, and block-based rearrangements, whose parameters scale with problem size.

In particular, Or–opt relocations [3], k–exchange moves [24], and cross–exchange operations [32] are applied using intensity–dependent parameterizations, while parameter-free operators such as double-bridge move [5,15] are employed at higher shaking levels. Additional variability is introduced through segment shuffling and block recombination mechanisms [28].

For segment-based operators, such as Or-opt [3] and cross-exchange [32], segment lengths are defined as bounded fractions of n, clipped between predefined minimum and maximum values. As a result, lower shaking levels modify short segments even in large instances, whereas higher levels operate on proportionally larger portions of the tour. Similarly, for exchange-based operators (e.g., k-exchange [24]), the number of exchanged nodes increases with n, while non-parametric operators such as double-bridge are applied without modification [5,15].

At each shake step, the neighborhood $\mathcal{N}_k(x)$ is implicitly defined by a set of perturbation operators $\mathcal{O}_k$ associated with the shaking level k (lines 3–11), whose parameters are scaled according to the size of the instance n (lines 1–2). The perturbation is then obtained by sampling an operator from $\mathcal{O}_k$ and applying it to the incumbent solution (line 14), with repeated attempts allowed to avoid trivial modifications (lines 12–18). The set of operators $\mathcal{O}_k$ is indexed by k to induce progressively stronger perturbations, ranging from mild structural changes to larger-scale tour rearrangements [5,15].

3.3 RL-Driven Local Search Procedure

The RLLocalSearch procedure implements the intensification phase of the proposed framework (line 11 of Algorithm 1). Starting from the shaken tour x', its objective is to exploit the local structure of the solution and produce an improved tour by sequentially applying local search operators. The selection of these operators is guided by a Q-learning mechanism combined with an ε-greedy policy, which allows adaptive control of the intensification process based on the observed effectiveness of operator transitions. The two components are independent: MARL (Sect. 3.1) operates only during construction with its own action-value estimates, while RL-GVNS maintains a separate Q-table over local search operator transitions.

Algorithm 3 describes the reinforcement learning–driven local search procedure used. The Q-table Q is initialized with zeros once before the outer GVNS loop (Algorithm 1) and is shared across all local search calls, allowing the agent to accumulate transition knowledge throughout the entire search. At the start of each call, the exploration rate is set to its current value ε, which decays after each call according to a multiplicative schedule (defined in Sect. 3.3), gradually shifting from exploration to exploitation.

The incumbent solution is set to the shaken tour x' and its objective value $c'' = f(x')$ is calculated (line 2). The set of operators is then initialized as $\mathcal{A} \leftarrow \mathcal{O}$ (line 4). The local search operates on a fixed set of neighborhood operators

$$\mathcal{O} = \left\{ \begin{array}{ll} \text{TwoOpt [9]}, & \text{OneMoveInsertion [11]}, \\ \text{ThreeOpt [6]}, & \text{DoubleBridge [15]} \end{array} \right\} \tag{2}$$

which differ in neighborhood size and structural impact. The initial operator is selected from $\mathcal{A}$ (line 4), and the local search proceeds while admissible operators remain (lines 6-19). At each iteration, the selected operator is applied to the

Algorithm 3 RL-driven Local Search

Require: Initial solution x', initial cost c', operator set $\mathcal{O}$, learning rate α, discount factor γ, initial exploration ε_0, minimum exploration $\varepsilon_{\min}$, exploration decay δ, max iterations $L_{\max}$, failure limit $F_{\max}$

Ensure: Improved solution x''

1: Q shared across calls (initialized once before GVNS loop); retrieve current ε
2: $x'' \leftarrow x'$, $c'' \leftarrow c'$
3: $\ell \leftarrow 0$, $f \leftarrow 0$ ▷ iteration and consecutive-failure counters
4: $\mathcal{A} \leftarrow \mathcal{O}$
5: $o \leftarrow \textsc{Select}(\mathcal{A}, Q, \varepsilon)$
6: **while** $\mathcal{A} \neq \emptyset$ **and** $\ell < L_{\max}$ **and** $f < F_{\max}$ **do**
7: $\quad \tilde{x} \leftarrow o(x'')$
8: $\quad \tilde{c} = f(\tilde{x})$
9: $\quad r \leftarrow \textsc{Reward}(c'', \tilde{c})$ ▷ reward design and Q-update
10: $\quad$ **if** $\tilde{c} < c''$ **then**
11: $\quad\quad x'' \leftarrow \tilde{x}$, $c'' \leftarrow \tilde{c}$
12: $\quad\quad \mathcal{A} \leftarrow \mathcal{O} \setminus \{o\}$ ▷ first-improvement acceptance
13: $\quad\quad f \leftarrow 0$ ▷ reset failure counter
14: $\quad$ **else**
15: $\quad\quad \mathcal{A} \leftarrow \mathcal{A} \setminus \{o\}$; $f \leftarrow f + 1$ ▷ operator pruning
16: $\quad$ **end if**
17: $\quad \ell \leftarrow \ell + 1$
18: $\quad$ Select next operator o from $\mathcal{A}$ using ε-greedy policy ▷ ε-greedy selection
19: **end while**
20: **return** x''

current solution, resulting in a candidate tour $\tilde{x}$ (line 7), which is subsequently evaluated to obtain its cost $\tilde{c}$ (line 8).

State–Action Model. Let $\mathcal{O} = \{o_1, \ldots, o_m\}$ denote the set of local search operators, and let $\tau : \mathcal{O} \to \{1, \ldots, m\}$ be an index map. The method maintains a Q-table $Q \in \mathbb{R}^{m \times m}$, where the state $s \in \{1, \ldots, m\}$ encodes the index of the previously applied operator and the action $a \in \{1, \ldots, m\}$ encodes the index of the next operator. Hence, $Q[s, a]$ estimates the utility of applying the operator a given that the previous operator was s. This operator-transition encoding is intentionally compact: it avoids the overhead of computing solution-quality indicators at every step, keeps the state space size fixed at m^2 regardless of instance size, and captures sequential operator dependencies that have been shown to influence local search effectiveness [10,36]. Richer state representations incorporating, e.g., current gap to best-known or stagnation counters, may improve learning precision but would increase both the state space and the tuning burden. This is left as a direction for future work.

Reward Design. After evaluating the candidate tour, the reward is calculated from the incumbent cost c'' and the candidate cost $\tilde{c}$ (line 8) as

$$r(c'', \tilde{c}) = \begin{cases} \dfrac{c'' - \tilde{c}}{\max(c'', 10^{-9})} & \text{if } \tilde{c} < c'', \\ -\lambda \cdot \dfrac{\tilde{c} - c''}{\max(c'', 10^{-9})} & \text{if } \tilde{c} > c'', \\ -0.01 & \text{if } \tilde{c} = c'', \end{cases} \tag{3}$$

where $\lambda \geq 1$ controls the asymmetry between positive and negative feedback: a cost reduction is rewarded in proportion to its relative magnitude, while a deterioration of equivalent size is penalized by λ. The small constant -0.01 assigned to neutral outcomes serves mainly as a tie-breaking signal, gently discouraging operators that produce no change without distorting the overall reward scale. After computing the reward, the Q-table is updated via the Bellman rule (Eq. 1), using the best estimated value for the successor state (line 9).

First-Improvement Acceptance and Operator Pruning. When the candidate tour improves on the incumbent ($\tilde{c} < c''$, line 10), the solution is immediately updated (line 11) and the admissible set is reset to $\mathcal{A} \leftarrow \mathcal{O} \setminus \{o\}$ (line 12), excluding the operator just applied so that the remaining operators can act on the improved solution. Conversely, if no improvement is found, the operator is dropped from $\mathcal{A}$ (line 15), avoiding repeated applications of a move that has already been tried without success in the current call.

ε-Greedy Operator Selection. After updating the admissible set, the next operator is selected from $\mathcal{A}$ according to an ε-greedy policy (line 18):

$$a \sim \begin{cases} \text{Uniform}(\mathcal{A}) & \text{with probability } \varepsilon, \\ \arg\max\limits_{a \in \mathcal{A}} Q[s, a] & \text{with probability } 1 - \varepsilon \end{cases} \tag{4}$$

which balances exploitation of high-utility operator transitions and exploration of alternative moves. The procedure ends when the admissible set becomes empty, returning the best solution found. After each call to the local search routine, the exploration rate is updated according to a multiplicative decay schedule,

$$\varepsilon \leftarrow \max(\varepsilon_{\min}, \varepsilon \cdot \delta) \tag{5}$$

with $\delta \in (0.0, 1.0)$, gradually shifting the policy from exploration to exploitation in successive calls.

4 Computational Experiments

The experimental design follows a 2×2 factorial structure, crossing two initializers (RCL or MARL) with two local search variants (fixed VND or RL-driven), yielding four configurations. This layout allows the contribution of each component to be assessed independently: varying the initializer while holding the local

search fixed measures the effect of the construction strategy, and varying the local search while holding the initializer fixed measures the effect of adaptive operator selection. All four configurations share the same adaptive shaking procedure (Algorithm 2). In standard GVNS, intensification follows a deterministic VND with the fixed operator order of Eq. 2. In RL-GVNS, it is replaced by the Q-learning-driven local search of Algorithm 3.

4.1 Methodology

All experiments were conducted on a single reference machine. The computational platform consisted of a MacBook Pro equipped with an Apple M3 Pro processor (11 cores) and 18 GB of RAM. All runs were executed in single-threaded mode, without parallelization. A representative subset of widely used TSPLIB (Traveling Salesman Problem Library) instances was selected to cover small, medium, and large-scale problem sizes, allowing a comprehensive evaluation of algorithmic performance. The corresponding instance characteristics and best known solution values reported in the literature are summarized in Table 2.

Table 2. TSPLIB instances used in this study, including problem size and best known solution values reported in the literature.

Instance	n	Best Known
bays29	29	2020
swiss42	42	1273
gr48	48	5046
berlin52	52	7542
kroA100	100	21282
kroB100	100	22141
ch150	150	6528
si175	175	21407
pr226	226	80369
pcb442	442	50778

For each algorithm–initialization pairing, defined by a specific search algorithm combined with a particular solution initialization strategy, 30 independent runs were conducted to account for stochastic variability. All methods used identical stopping criteria based on two conditions: a maximum number of iterations and a limit on consecutive non-improving iterations. For instances with at most 100 cities, the maximum number of iterations was set to 500 and the non-improvement limit to 200 consecutive iterations. For medium-sized instances ($100 < n < 200$) and larger instances ($n \geq 200$), these limits were reduced to 300 total iterations and 100 consecutive non-improving iterations, respectively. Performance was primarily evaluated on the final tour length, while solution stability across independent runs was also considered.

RL Parameters. Parameter values were set by grid search on three held-out instances (`berlin52`, `kroA100`, `pr226`), chosen to represent small, medium, and large problem scales. Candidate grids were built around ranges reported in the literature [1,36]: $\alpha \in \{0.10, 0.22, 0.30, 0.50\}$, $\gamma \in \{0.30, 0.50, 0.70, 0.80\}$, $\varepsilon_0 \in \{0.40, 0.60, 0.80\}$, $\delta \in \{0.95, 0.98, 0.995\}$, $\varepsilon_{\min} \in \{0.02, 0.05\}$, and $\lambda \in \{1.0, 1.2, 1.4, 2.0\}$. Rather than optimizing all parameters jointly, we followed the sequential protocol of [1], locking each group once selected before moving to the next. Every candidate setting was evaluated over 10 runs per calibration instance; the setting with the lowest mean gap was kept. A notable outcome was that the best values differ noticeably between small ($n \leq 100$) and larger instances ($n > 100$), which led to adopting separate parameter schedules for each regime. For the MARL initializer, the softmax temperature was raised above the value used in [1] to sharpen action selection, and the number of learning iterations was reduced for larger instances where additional iterations offered no further improvement. The resulting values are listed in Table 3.

Table 3. Parameters used in the proposed framework.

Parameter	Value(s)	Defined
Learning rate (α)	0.30 (small), 0.22 (large)	Algorithm 3, Eq. 1
Discount factor (γ)	0.50 (small), 0.80 (large)	Algorithm 3, Eq. 1
Initial exploration rate (ε_0)	0.60 (small), 0.40 (large)	Algorithm 3
Exploration decay (δ)	0.98 (small), 0.995 (large)	Algorithm 3
Minimum exploration ($\varepsilon_{\min}$)	0.05 (small), 0.02 (large)	Algorithm 3
Penalty scale (λ)	1.2 (small), 1.4 (large)	Eq. 3
Failure limit	10 consecutive failures	Algorithm 3
Max. local search iterations	300 (small), 600 (large)	Algorithm 3
Softmax temperature (β)	3	Sect. 3.1
MARL iterations	10,000 ($n < 100$), 5,000 ($n > 100$)	Sect. 3.1

Analysis. Table 4 reports the results obtained over 30 independent runs for each algorithm–initialization configuration. For each instance, *Best* denotes the best tour cost achieved, while *Max* and *Mean* correspond to the worst and average objective values, respectively. The *Gap (%)* represents the mean relative deviation from the best-known solution (BKS), calculated as:

$$\text{Gap}(\%) = \frac{1}{R} \sum_{i=1}^{R} \frac{c_i - \text{BKS}}{\text{BKS}} \times 100, \tag{6}$$

where c_i is the cost obtained in the i-th run and $R = 30$. The *BKSH* column reports the number of runs that reach the BKS. The *IterOpt* metric indicates the average number of iterations required to reach the best solution, and *Time (s)* denotes the mean execution time. For each instance, values highlighted in bold indicate the best-performing configuration for the corresponding metric.

Table 4. Performance results over 30 independent runs for each algorithmic configuration.

Instance	Alg.	Best	Max	Mean	Gap (%)	BKSH	IterOpt	Time (s)
bays29	(1)	**2020**	**2020**	**2020**	**0.00**	**30**	3.13	1.17
	(2)	**2020**	**2020**	**2020**	**0.00**	**30**	3.07	1.02
	(3)	**2020**	**2020**	**2020**	**0.00**	**30**	3.60	1.41
	(4)	**2020**	**2020**	**2020**	**0.00**	**30**	**2.60**	**0.91**
swiss42	(1)	**1273**	**1273**	**1273**	**0.00**	**30**	1.90	1.12
	(2)	**1273**	**1273**	**1273**	**0.00**	**30**	0.60	**0.10**
	(3)	**1273**	**1273**	**1273**	**0.00**	**30**	1.87	1.62
	(4)	**1273**	**1273**	**1273**	**0.00**	**30**	**0.50**	0.14
gr48	(1)	**5046**	**5046**	**5046**	**0.00**	**30**	9.47	11.73
	(2)	**5046**	**5046**	**5046**	**0.00**	**30**	**6.70**	**8.37**
	(3)	**5046**	**5046**	**5046**	**0.00**	**30**	11.93	17.80
	(4)	**5046**	**5046**	**5046**	**0.00**	**30**	9.57	12.27
berlin52	(1)	**7542**	**7542**	**7542**	**0.00**	**30**	3.07	3.38
	(2)	**7542**	**7542**	**7542**	**0.00**	**30**	1.47	**1.18**
	(3)	**7542**	**7542**	**7542**	**0.00**	**30**	3.23	5.25
	(4)	**7542**	**7542**	**7542**	**0.00**	**30**	**1.33**	1.56
kroA100	(1)	**21282**	**21282**	**21282**	**0.00**	**30**	15.87	73.07
	(2)	**21282**	**21282**	**21282**	**0.00**	**30**	8.20	**35.91**
	(3)	**21282**	**21282**	**21282**	**0.00**	**30**	17.37	120.76
	(4)	**21282**	**21282**	**21282**	**0.00**	**30**	**6.70**	48.44
kroB100	(1)	**22141**	22199	22162.27	0.10	19	**59**	683.12
	(2)	**22141**	22199	22150.67	0.04	25	75.16	**503.27**
	(3)	**22141**	22199	22144.87	0.02	28	71.54	1026.63
	(4)	**22141**	**22141**	**22141**	**0.00**	**30**	110.13	1212.83
ch150	(1)	**6528**	6564	6532.83	0.07	19	173.23	1967.42
	(2)	**6528**	6564	6534.70	0.10	21	125.00	1484.35
	(3)	**6528**	6566	6535.17	0.11	20	**113.74**	**1011.45**
	(4)	**6528**	**6549**	**6530.77**	**0.04**	**26**	131.33	1327.90
si175	(1)	**21407**	21421	21409.57	0.01	15	102.82	3387.24
	(2)	**21407**	21420	**21408.00**	**0.01**	17	180.00	3324.80
	(3)	**21407**	21427	21410.43	0.02	17	115.20	1204.38
	(4)	**21407**	**21419**	21410.89	**0.01**	**20**	171.00	**1067.30**
pr226	(1)	**80369**	80442	80381.00	**0.01**	27	83.00	5726.51
	(2)	**80369**	80442	80421.00	**0.01**	28	83.00	3512.51
	(3)	**80369**	80440	80371.37	**0.00**	**29**	**79.45**	2000.05
	(4)	**80369**	**80373**	**80369.13**	**0.00**	**29**	92.55	**1573.50**
pcb442	(1)	**50778**	51210	50950.00	0.10	18	135.00	5200.00
	(2)	**50778**	51150	50910.00	0.10	21	120.00	4800.00
	(3)	**50778**	51180	50930.00	0.07	23	110.00	6100.00
	(4)	**50778**	**50980**	**50850.00**	**0.06**	**27**	**95.00**	**4200.00**

The algorithmic configurations evaluated are: (1) RCL + GVNS, (2) MARL + GVNS, (3) RCL + RL-GVNS, and (4) MARL + RL-GVNS.

On small-scale instances, all four configurations reach the best-known solution in every run, and the differences between them are limited to convergence speed rather than solution quality. On `bays29` and `swiss42`, for example, all variants achieve a zero average gap and perfect BKSH rates, with only minor variation in iteration counts and runtimes. The picture changes as instance size increases. On

medium and large instances, the adaptive configurations produce more consistent results across independent runs, with lower mean gaps and higher best-known solution hit rates than the non-adaptive alternatives.

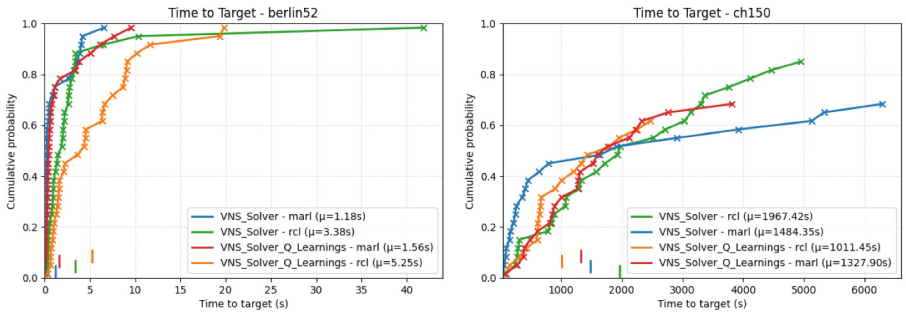
The runtime behavior is also noteworthy. Although the added complexity of MARL initialization and RL-guided local search might be expected to increase computation time, this is not consistently observed. For most instances, adaptive configurations reach high-quality solutions at least as fast as their non-adaptive counterparts, and in some cases faster. Better-informed operator sequencing appears to reduce unproductive search effort near high-quality regions of the landscape, where fixed operator orderings tend to lose effectiveness.

An exception is observed on `kroB100`, where the MARL + RL-GVNS configuration incurs a higher computational time (1 213 s) compared to its non-adaptive counterpart (503 s). However, this additional cost is accompanied by a marked improvement in robustness, as the adaptive variant is the only configuration to consistently reach the best-known solution in all 30 runs, yielding a zero average gap. In contrast, faster non-adaptive configurations exhibit residual gaps and lower BKSH rates. A similar pattern is observed on larger instances, such as `ch150` and `si175`, although with distinct trade-offs. For `ch150`, MARL-based configurations achieve lower average gaps (down to 0.04%) while also showing reduced intensification times and higher best-known solution hit rates, indicating that adaptive mechanisms improve both efficiency and solution quality. The same pattern is observed in `si175`.

As shown in Figs. 1a and 1b, all configurations eventually reach the best–known solution. However, the Empirical Cumulative Distribution Functions (ECDFs) reveal clear differences in the convergence dynamics. Learning–based configurations, particularly under MARL initialization, exhibit steeper early curves and a noticeable leftward shift, indicating a higher probability of reaching the target solution at earlier stages. In contrast, RCL–based variants show more gradual convergence, even when combined with learning. This effect becomes more pronounced in harder instances, where adaptive configurations converge faster and more consistently, in agreement with the lower average gaps and higher BKSH rates reported in Table 4.

To assess whether the observed performance differences are statistically significant, two complementary non–parametric analyses were performed following the framework of [8]. The Friedman test applied to mean tour costs across all 10 instances [8] produced $\chi^2 = 3.48$ ($p = 0.323$), which is not statistically significant. The five trivial instances (`bays29`, `swiss42`, `gr48`, `berlin52`, `kroA100`) yield tied ranks for all configurations, and restricting the analysis to the five hard instances produces the same statistic, as inconsistent behavior across instances (e.g., `si175`) prevents global significance. Average ranks favor the combined adaptive configuration: (4) MARL+RL–GVNS = 2.05, (2) MARL+GVNS = 2.55, (3) RCL+RL-GVNS = 2.65, and (1) RCL+GVNS = 2.75.

Pairwise Mann–Whitney U tests with Holm–Bonferroni correction [14] were applied to the five hard instances using run distributions derived from BKSH counts via two–point (BKS / max) imputation. Significant outcomes were found



(a) `berlin52` - All Combinations **(b)** `ch150` - All Combinations

Fig. 1. Time-to-target ECDF for representative TSPLIB instances.

for `kroB100` and `pcb442`. For `pcb442`, MARL+RL–GVNS (4) outperforms RCL+GVNS (1) with a medium–sized effect ($U = 603$, $p_{\text{adj}} = 0.0170$, $\hat{A}_{12} = 0.670$), confirming a statistically significant robustness advantage for the combined adaptive configuration. By contrast, the RCL+RL–GVNS (3) vs. MARL+RL–GVNS (4) comparison yields no significant difference ($U = 520$, $p_{\text{adj}} = 0.2546$, $\hat{A}_{12} = 0.578$), indicating that the adaptive local search alone may be insufficient without the MARL initializer on the hardest instances.

5 Conclusion

Classical GVNS applies local search operators in a fixed order regardless of how the search is evolving, a behavior that tends to become progressively less efficient as the algorithm approaches high-quality regions of the solution space. This work addressed that limitation by embedding a Q-learning mechanism directly into the intensification phase and pairing it with a MARL-based initialization that acts independently at the construction stage.

On small TSPLIB instances, all four configurations converge to the best-known solution in every run and the differences in runtime and iteration count are negligible. The more informative comparison comes from medium and large instances, where a static operator order begins to affect both solution quality and convergence consistency. The MARL + RL-GVNS configuration consistently produced lower average gaps and higher best-known solution hit rates, and in several cases it reached high-quality solutions faster than the non-adaptive variants. The `kroB100` instance illustrates a common pattern well. The adaptive configuration required roughly twice the wall-clock time of the fastest non-adaptive variant, yet it was the only one to reach the best-known solution in all 30 independent runs. The robustness advantage is substantial and repeatable across trials. Non-parametric statistical tests corroborate these observations: the Friedman test over all instances did not reach significance, largely because the five small instances produce tied ranks across configurations, but

pairwise tests on the harder instances confirmed statistically significant differences favoring MARL+RL-GVNS, with a medium-sized effect on `kroB100` and `pcb442`. Notably, the adaptive local search alone was not sufficient to produce a significant improvement on the hardest instances without the MARL initializer, suggesting that the two components are complementary rather than interchangeable.

The adaptive shaking strategy is shared by all four configurations and already improves upon a standard fixed-perturbation scheme. The RL components build on that foundation and do not replace it. The primary limitation of the current approach lies in the state representation, which encodes only the most recently applied operator. Extending it to include, for instance, the current gap relative to the best-known solution or a count of non-improving iterations would give the policy a richer basis for decisions, though at the cost of a larger Q-table and additional parameter tuning. Evaluating the framework on larger benchmark instances and adapting it to other neighborhood-based metaheuristics are the most natural directions for future work.

Acknowledgments. This work is partially supported by CAPES (Grant number #001) and FAPES (#2026-6NX7B).

Declaration of AI-Assisted Technologies. The use of the tool (Gemini - Google) was strictly limited to writing support and did not involve the generation of scientific content, results, or analysis. All AI-assisted text was subsequently reviewed and revised by the authors, who take full responsibility for the accuracy, integrity, and originality of the final work.

References

1. Alipour, M.M., Razavi, S.N.: A new multiagent reinforcement learning algorithm to solve the symmetric traveling salesman problem. Multiagent Grid Syst. **11**(2), 107–119 (2015)
2. Alipour, M.M., Razavi, S.N., Feizi Derakhshi, M.R., Balafar, M.A.: A hybrid algorithm using a genetic algorithm and multiagent reinforcement learning heuristic to solve the traveling salesman problem. Neural Comput. Appl. **30**(9), 2935–2951 (2018)
3. Babin, G., Deneault, S., Laporte, G.: Improvements to the or-opt heuristic for the symmetric travelling salesman problem. J. Oper. Res. Soc. **58**(3), 402–407 (2007)
4. Basu, S., Ghosh, D.: A review of the tabu search literature on traveling salesman problems. In: IIMA Working Paper WP2008-10-01. Indian Institute of Management Ahmedabad, Research and Publication Department (2008)
5. Bentley, J.L.: Fast algorithms for geometric traveling salesman problems. ORSA J. Comput. **4**(4), 387–411 (1992)
6. Blazinskas, A., Misevicius, A.: Combining 2-opt, 3-opt and 4-opt with k-swap-kick perturbations for the traveling salesman problem. In: Proceedings of the 17th International Conference on Information and Software Technologies (IT 2011), pp. 1–6. Kaunas, Lithuania (2011)
7. da Silva, R.F., Urrutia, S.: A general VNS heuristic for the traveling salesman problem with time windows. Discret. Optim. **7**(4), 203–211 (2010)

8. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. J. Mach. Learn. Res. **7**, 1–30 (2006)
9. Feo, T.A., Resende, M.G.C.: Greedy randomized adaptive search procedures. J. Global Optim. **6**(2), 109–133 (1995)
10. Fialho, A., Da Costa, L., Schoenauer, M., Sebag, M.: Analyzing bandit-based adaptive operator selection mechanisms. Ann. Math. Artif. Intell. **60**(1), 25–64 (2010)
11. Glover, F., Rego, C.: Local search and metaheuristics for the traveling salesman problem. In: Gutin, G., Punnen, A. (eds.) The Traveling Salesman Problem and Its Variations, pp. 309–368. Kluwer Academic Publishers, Boston (2002)
12. Hansen, P., Mladenović, N., Moreno Pérez, J.A.: Variable neighbourhood search: methods and applications. 4OR **6**(4), 319–360 (2008)
13. Hansen, P., Mladenović, N., Todosijević, R., Hanafi, S.: Variable neighborhood search: basics and variants. Euro J. Comput. Optim. **5**(3), 423–454 (2017)
14. Holm, S.: A simple sequentially rejective multiple test procedure. Scand. J. Stat. **6**(2), 65–70 (1979)
15. Hoos, H.H., Stützle, T.: Stochastic local search. In: Handbook of Approximation Algorithms and Metaheuristics, pp. 297–307. Chapman and Hall/CRC (2018)
16. Hore, S., Chatterjee, A., Dewanji, A.: Improving variable neighborhood search to solve the traveling salesman problem. Appl. Soft Comput. **68**, 83–91 (2018)
17. Jeong, H.Y., Song, B.D.: Meta-learning-based adaptive operator selection for traveling salesman problem. Appl. Soft Comput. **185**, 113930 (2025)
18. Karakostas, P., Panoskaltsis, N., Mantalaris, A., Georgiadis, M.C.: Optimization of CAR T-cell therapies supply chains. Comput. Chem. Eng. **139**, 106913 (2020)
19. Karakostas, P., Sifaleras, A.: A double-adaptive general variable neighborhood search algorithm for the solution of the traveling salesman problem. Appl. Soft Comput. **121**, 108746 (2022)
20. Karakostas, P., Sifaleras, A., Georgiadis, M.C.: A general variable neighborhood search-based solution approach for the location-inventory-routing problem with distribution outsourcing. Comput. Chem. Eng. **126**, 263–279 (2019)
21. Karimi-Mamaghan, M., Mohammadi, M., Meyer, P., Karimi-Mamaghan, A.M., Talbi, E.G.: Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. Eur. J. Oper. Res. **296**(2), 393–422 (2022)
22. Kool, W., van Hoof, H., Welling, M.: Attention, learn to solve routing problems! In: International Conference on Learning Representations (2019)
23. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B.: The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. Wiley, Chichester (1985)
24. Marx, D.: Searching the k-change neighborhood for tsp is W[1]-hard. Oper. Res. Lett. **36**(1), 31–36 (2008)
25. Menéndez, B., Bustillo, M., Pardo, E.G., Duarte, A.: General variable neighborhood search for the order batching and sequencing problem. Eur. J. Oper. Res. **263**(1), 82–93 (2017)
26. Mladenović, N., Hansen, P.: Variable neighborhood search. Comput. Oper. Res. **24**(11), 1097–1100 (1997), citeSeerX: 10.1.1.800.1797
27. Osman, I.H., Kelly, J.P.: Meta-heuristics: an overview. In: Osman, I.H., Kelly, J.P. (eds.) Meta-Heuristics: Theory and Applications, pp. 1–21. Kluwer Academic Publishers, Boston (1996)
28. Papalitsas, C., Karakostas, P., Andronikos, T.: A performance study of the impact of different perturbation methods on the efficiency of GVNS for solving TSP. Appl. Syst. Innov. **2**, 31 (2019)

29. Queiroz dos Santos, J.P., de Melo, J.D., Duarte Neto, A.D., Aloise, D.: Reactive search strategies using reinforcement learning, local search algorithms and variable neighborhood search. Expert Syst. Appl. **41**(10), 4939–4949 (2014)
30. Ruan, Y., Cai, W., Wang, J.: Combining reinforcement learning algorithm and genetic algorithm to solve the traveling salesman problem. J. Eng. **2024**, 2051–3305 (2024)
31. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction, Adaptive Computation and Machine Learning. MIT Press, Cambridge (1998)
32. Taillard, E., Badeau, P., Gendreau, M., Guertin, F., Potvin, J.Y.: A tabu search heuristic for the vehicle routing problem with soft time windows. Transp. Sci. **31**(2), 170–186 (1997)
33. Todosijević, R., Mladenović, M., Hanafi, S., Mladenović, N., Crévits, I.: Adaptive general variable neighborhood search heuristics for solving the unit commitment problem. Int. J. Electr. Power Energy Syst. **78**, 873–883 (2016)
34. Venkatesh, P., Srivastava, G., Singh, A.: A general variable neighborhood search algorithm for the k-traveling salesman problem. Procedia Comput. Sci. **143**, 189–196 (2018), 8th International Conference on Advances in Computing & Communications (ICACC-2018)
35. Watkins, C.J.C.H.: Learning from Delayed Rewards. Ph.D. thesis, University of Cambridge, Cambridge (1989)
36. Wauters, T., Verbeeck, K., De Causmaecker, P., Vanden Berghe, G.: Boosting metaheuristic search using reinforcement learning. In: Talbi, E.G. (ed.) Hybrid Metaheuristics, pp. 433–452. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-30671-6_17