An Efficient Adaptive Large Neighborhood Search Algorithm for the Flying Sidekick Traveling Salesman Problem

Duc D. Vu[a], Le Thi Hong[b], Tran Nam Khanh[c,e], Nguyen Dinh Cong[b], Vu Duc Minh[d,e,*]

[a]*School of Management, University of Michigan-Flint, 303 East Kearsley Street, Flint, MI, 48502-1950, USA*
[b]*Hong Duc University, 565 Quang Trung, Hac Thanh, Thanh Hoa, Viet Nam*
[c]*Hanoi University of Science, 334 Nguyen Trai, Thanh Xuan, Ha noi, Viet Nam*
[d]*National Economics University, 207 Giai Phong, Hai Ba Trung, Ha noi, Viet Nam*
[e]*Smart Logistics and Supply Chain Management Laboratory, National Economics University, Viet Nam*

Abstract

This paper investigates the Flying Sidekick Traveling Salesman Problem (FSTSP), and proposes an efficient Adaptive Large Neighborhood Search (ALNS) algorithm. Our proposed framework operates directly on a complete solution representation, eliminating the reconstruction step required by indirect encodings and making temporal information immediately accessible during the search process. A stage-based mechanism is introduced to efficiently update drone sorties while preserving the temporal consistency of drone operations when modifications are made to the truck route. In addition, destroy and repair operators are organized into coherent operator groups to avoid inconsistent intermediate solutions and improve large neighborhood exploration. The proposed method further incorporates six problem-specific local search moves and derives constant-time move evaluation formulas, with local search supported by an adaptation of the Static Move Descriptor (SMD) technique to accelerate the computational process. Experiments demonstrate that the proposed ALNS outperforms several

*Corresponding author
Email address: minhvd@neu.edu.vn (Vu Duc Minh)

state-of-the-art algorithms for the FSTSP, while maintaining stable performance on large-scale instances with up to 500 customers within 1800 seconds.

Keywords: Flying Sidekick Traveling Salesman Problem, Adaptive Large Neighborhood Search, Static Move Descriptor, Discrete Optimization

1. Introduction

The last mile, i.e., the final stage of the supply chain in which goods are transported from a distribution facility to end customers, has become one of the most operationally challenging segments of modern logistics. This stage is marked by dispersed delivery destinations, fluctuating demand, urban congestion with associated concerns over air quality (Nguyen et al., 2022), and rising customer expectations. The rapid expansion of e-commerce has further intensified pressure on logistics providers to improve efficiency, responsiveness, and service quality in this critical segment.

To address these challenges, both researchers and practitioners have increasingly examined the role of emerging technologies in last-mile operations. Firms have begun integrating unmanned aerial vehicles, or drones, into their logistics networks, delivering even time-sensitive or perishable products such as food and medicines. For instance, Wendy noted that "Drone delivery is kind of a natural extension of the delivery platform" [1]. This momentum is reflected in market forecasts, with the global drone delivery market expected to reach $28 billion by 2028, signaling strong confidence in this technology's future role in logistics.

Nevertheless, significant barriers remain. Drones continue to face technical constraints, particularly in payload capacity and flight endurance. In addition, fully drone-based systems require considerable infrastructure investment and must comply with evolving regulatory requirements. Consequently, hybrid delivery systems in which drones complement conventional trucks have emerged as a practical alternative. By combining the flexibility

[1] https://www.fiercesensors.com/electronics/wendys-beefs-drone-delivery-tech-cooks

of drones with established infrastructure and capacity of truck delivery, such systems can capitalize on the advantages of both delivery modes.

In this paper, we study the Flying Sidekick Traveling Salesman Problem (FSTSP) (Murray and Chu, 2015), where a truck and a drone works in tandem to deliver packages to customers and propose an Adaptive Large Neighborhood Search (ALNS) solution for this problem. Our main contributions are as follows.

- We develop an ALNS framework that operates directly on a full solution representation, avoiding the reconstruction required by indirect encodings and allowing the temporal information required by the operators to be immediately available.
- We introduce a stage-based mechanism to update drone sorties when the truck route changes, preserving the temporal structure of drone operations while allowing the truck route to be modified efficiently.
- We organize destroy and repair operators into coherent group operators, avoiding inconsistent intermediate solutions and exploring large neighborhoods in a more controlled manner.
- We design a set of six problem-specific local search neighborhoods for the FSTSP, providing a more targeted intensification mechanism than generic routing moves.
- We derive constant-time move evaluation formulas for all proposed neighborhoods and adapt the Static Move Descriptor (SMD) technique to accelerate the computational process.
- The proposed ALNS is robust and competitive with state-of-the-art algorithms for the FSTSP, including MD–SPP–H (Schaumann et al., 2025), HGA–TAC (Mahmoudinazlou and Kwon, 2024), EP–All (Agatz et al., 2018), and SPP–All (Kundu et al., 2022), on a benchmark set of 150 instances. The computational results show stable performance on additional 84 large-scale instances with up to 500 customers within

a time limit of 1800 seconds. We also discuss the robustness of our proposed solution.

The remainder of the paper is organized as follows. Section 2 reviews related literature, while Section 3 formally defines our problem setting. Section 4 presents our ALNS framework, including the proposed destroy-repair group operators and local search with the SMD. Section 5 reports the computational results of the proposed solution against state-of-the art algorithms and its effectiveness. Section 6 concludes our paper.

2. Related work

Truck–drone collaborative routing has received increasing attention in last-mile delivery. The FSTSP introduced by Murray and Chu (2015) extends the classical TSP by allowing a truck and a drone to serve customers cooperatively, with each customer visited exactly once by either vehicle. Existing solution approaches include heuristics and metaheuristics such as GRASP, variable neighborhood search, hybrid genetic algorithms, and dynamic programming-based methods (Ha et al., 2018; De Freitas and Penna, 2020; Agatz et al., 2018; Mahmoudinazlou and Kwon, 2024), as well as exact methods based on branch-and-bound, branch-and-price, column-and-row generation, and branch-and-cut (Bouman et al., 2018; Poikonen et al., 2019; Roberti and Ruthmair, 2021; Boccia et al., 2021, 2023). Comprehensive surveys of truck–drone routing and drone delivery applications can be found in Boysen et al. (2020), Li et al. (2021), Macrina et al. (2020), Moshref-Javadi and Winkenbach (2021), and Moradi et al. (2024).

Despite these advances, solving large FSTSP instances remains challenging because truck-route decisions and drone-sortie decisions are strongly interdependent. Recent algorithms such as HGA–TAC, EP–All, SPP–All, and MD–SPP–H provide strong reference points for computational comparison (Mahmoudinazlou and Kwon, 2024; Agatz et al., 2018; Kundu et al., 2022; Schaumann et al., 2025). Closely related is the TSP-D of Agatz et al. (2018), where revisits and identical launch–recovery locations may be allowed. Motivated by the success of ALNS in large-scale routing problems (Ropke and

Pisinger, 2006) and truck–drone variants such as the TSP-D with multiple drops (Mara et al., 2022), we develop an ALNS framework that works directly on a full truck–drone representation and incorporates stage-based sortie updates, grouped destroy–repair operators, problem-specific local search, and Static Move Descriptors.

3. Problem statement

FSTSP can be modeled by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_0, v_1, ...v_{N-1}\}$ is a set of N nodes on the graph. Within $\mathcal{V}$, v_0 and v_{N-1} represent the depot, while the remaining are customers who need to be served by either the truck or the drone. $\mathcal{E}$ represents possible travel opportunities between two nodes i, j. For each arc $(i, j) \in \mathcal{E}$, we denote τ_{ij} and τ'_{ij} the truck and drone travel time between nodes i and j, respectively.

In FSTSP, only one truck and one drone are operating throughout the process. The movement of the truck is defined as a sequence of stages, where two consecutive stages correspond to the segment between either two consecutive customer visits or a customer and the depot. Formally, the truck route is represented by $\mathcal{C} = \{c_1, c_2, ..., c_K\}$, with $c_1 = v_0$ and $c_K = v_{N-1}$, where K is the number of locations visited by the truck. In any stage (except for the last one), the drone can be launched to serve customers. A drone sortie is composed of three components (c_1, c_2, c_3): the launch node c_1, the customer served by the drone c_2, and the recovery node c_3. Specifically, the drone trip can be represented as $\mathcal{D} = \{(c_{11}, c_{12}, c_{13}), (c_{21}, c_{22}, c_{23}), ..., (c_{D1}, c_{D2}, c_{D3})\}$, where D is the number of sorties and where each triple corresponds to the three components mentioned.

A solution to FSTSP is given by the truck schedule $\mathcal{C}$ along with a list of sorties carried out by the drone $\mathcal{D}$. The objective of FSTSP is to minimize *makespan*, which is the time that all customers are served and both the truck and the drone have returned to the depot. Figure 1 illustrates a scheduling solution for a FSTSP instance with 5 customers. The schedule starts and ends at the depot represented by v_0 and v_6, with truck delivery for customers 1-3 and drone delivery for customers 4 and 5 (i.e., $\mathcal{C} = \{v_0, v_1, v_2, v_3, v_6\}, \mathcal{D} =$

$\{(v_0, v_4, v_1), (v_2, v_5, v_6)\}$).

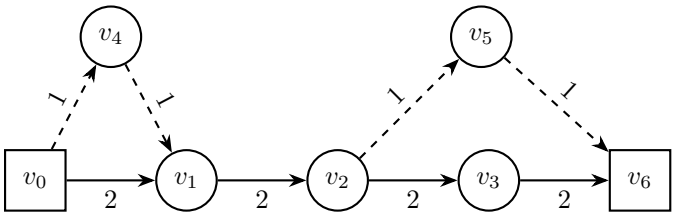

Figure 1: Example operation with five truck nodes $\{v_0(c_1), v_1(c_2), v_2(c_3), v_3(c_4), v_6(c_5)\}$ and two drone nodes $\{v_4,\ v_5\}$. The depot at $\{v_0,\ v_6\}$ is represented by a square. Solid arcs denote truck travel with distance 2; dashed arcs denote drone trips with distance 1. The objective value of this sample solution is 8.

We have the following assumptions:

1. A customer can only be visited once, by either the truck or the drone.
2. The truck has no capacity limit, while the drone is restricted. For each sortie, the drone can deliver to only one customer. After the drone is launched, the truck can continue to visit other customers as planned.
3. The launch and recovery nodes should be different, which means that the truck cannot stay at the same place where the drone is launched to wait for its recovery. In other words, loop operations are not allowed.
4. The drone has an endurance limit and other operational constraints. Once deployed, it must remain airborne and cannot land temporarily to conserve battery power, even when it is waiting for the truck at the recovery node. If the truck visits the recovery node first, the truck must wait for the drone's arrival and vice versa. Given the drone endurance limit DTL, a sortie (c_{d1}, c_{d2}, c_{d3}) with a launch and recovery stage l_d, r_d should satisfy:

$$\max\{\sum_{k=l_d}^{r_d-1} \tau_{c_k c_{k+1}}, \tau'_{c_{d1}\, c_{d2}} + \tau'_{c_{d2}\, c_{d3}}\} \leq DTL \tag{1}$$

where $\sum_{k=l_d}^{r_d-1} \tau_{c_k c_{k+1}}$ is the total truck travel time from launch to recovery stages and $\tau'_{c_{d1}\, c_{d2}} + \tau'_{c_{d2}\, c_{d3}}$ is drone travel time in sortie d.

4. Adaptive Large Neighborhood Search with Static Move Guidance

This section presents our ALNS-based solution approach, developed from the framework of Ropke and Pisinger (2006). Before describing its components in detail, we summarize the main algorithmic contributions below.

- We develop an ALNS framework that operates directly on a full solution representation $(\mathcal{C}, \mathcal{D})$, avoiding reconstruction and making temporal information immediately available to the operators.
- We introduce a stage-based mechanism for updating drone sorties after truck-route changes by recording launch and recovery stages to speed up computation
- We organize destroy and repair operators into coherent groups to ensure compatible combinations are applied, improving control over truck–drone feasibility interactions.
- We develop six specific local search neighborhoods to intensify the search over truck–drone interactions.
- We derive O(1) move evaluation formulas for proposed moves and use a lightweight SMD variant to accelerate the most expensive move.

Algorithm 1 summarizes the proposed ALNS framework. The algorithm starts by constructing an initial solution from the input instance I. This solution is used both as the current solution s and as the incumbent best feasible solution s^*. The weights of the destroy operators, repair operators, and group operators are then initialized before the main search loop begins.

At each iteration, the algorithm selects a group operator $G \in \mathcal{G}$ according to its adaptive weight, then chooses a compatible destroy operator $q \in G$ and repair operator $r \in G$ to generate a candidate solution $s' = r(q(s))$. This destroy–repair phase provides diversification, while the subsequent local search intensifies s' over problem-specific truck–drone neighborhoods. Temporary infeasibility is handled through the penalized objective value.

Algorithm 1 Adaptive Large Neighborhood Search for Flying Sidekick Traveling Salesman Problem

Require: Instance I, ALNS parameters
Ensure: A best feasible solution s^*

#Block 1: Initialize the initial solution and parameters
$s*, s \leftarrow$ Initial solution from I
Initialize weights ω for operators and group operators
while *stopping condition is not met* **do**
    #Block 2: Apply destroy–repair operators and local search operators
    Select a group operator G from the set $\mathcal{G}$
    Select a destroy operator $q \in G$ and a repair operator $r \in G$
    $s' \leftarrow r(q(s))$
    Apply local search operators on s' to find improved solutions
    #Block 3: Solution and parameter updates
    if $s'.\text{cost} \leq s.\text{cost}$ or $\mathcal{U}(0,1) < \exp\left(\frac{-(s'.\text{cost}-s.\text{cost})}{B \cdot Z \cdot T_c}\right)$ **then**
        $s \leftarrow s'$
        if s' is feasible and $s'.\text{cost} < s^*.\text{cost}$ **then**
            $s^* \leftarrow s'$
        end if
    end if
    Update weights ω and acceptance temperature T_c
end while
return s^*

The candidate s' is then evaluated by a simulated annealing-inspired acceptance rule. If s' is better than the current solution s, it is accepted directly; otherwise, it is accepted with probability $\exp\left(\frac{-(s'.\text{cost}-s.\text{cost})}{BZ \cdot T_c}\right)$, where BZ is the Boltzmann scaling factor and T_c is the current temperature. If accepted, s' replaces s, and s^* is updated only when s' is feasible and improves the incumbent. The operator weights and temperature are then updated, and the search continues until the stopping condition is met.

4.1. Objective evaluation

Auxiliary Notation. To express timing relations and objective components in a compact form, we introduce the following auxiliary notation. Consider a solution represented by a truck route $\mathcal{C} = \{c_1, c_2, ..., c_K\}$ and a drone trip

$\mathcal{D} = \{(c_{11}, c_{12}, c_{13}), (c_{21}, c_{22}, c_{23}), ..., (c_{D1}, c_{D2}, c_{D3})\}$.

1. $\mathcal{L} = \{l_1, l_2, \ldots, l_D\}$ denotes the set of launch stages from the drone sorties, where l_d is the stage index in $\mathcal{C}$ from which sortie d departs. In other words, $c_{l_k} = c_{k1}$, for $k = 1...D$.
2. $\mathcal{R} = \{r_1, r_2, \ldots, r_D\}$ denotes the set of recovery stages of the sorties, where r_d is the stage that the drone is recovered at sortie d. In other words, $c_{r_k} = c_{k3}$, for $k = 1...D$. Notice that $l_d < r_d, \forall d = 1 \ldots D$.
3. For the truck route $\mathcal{C}$, we define prefix travel times θ_{1j} for all $j = 1 \ldots K$, where θ_{1j} is the cumulative truck travel time from stage 1 to stage j. The travel time between any two stages $i < j$ is obtained in constant time by $\theta_{ij} = \theta_{1j} - \theta_{1i}$.
4. θ'_d denotes the *drone travel duration* of sortie $d = (c_{d1}, c_{d2}, c_{d3})$, computed as $\theta'_d = \tau'_{c_{d1}\, c_{d2}} + \tau'_{c_{d2}\, c_{d3}}$. This term presents just the flight time of the sortie. Any waiting time that arises when the drone reaches the recovery node earlier than the truck is not included in θ'_d and will be separately accounted for in the objective.

Using Figure 1 as example, the launch stages are $\mathcal{L} = \{1, 3\}$ and recovery stages are $\mathcal{R} = \{2, 5\}$. The cumulative truck travel times corresponding to its schedule are $\theta = \{0, 2, 4, 6, 8\}$, and drone travel duration are $\theta' = \{2, 2\}$.

The objective of a solution is defined as the sum of the truck travel time plus all synchronization and endurance penalties. Formally, we compute

$$\underbrace{\sum_{i=1}^{K-1} \tau_{c_i\, c_{i+1}}}_{\text{truck travel time}} + \underbrace{\sum_{d \in \mathcal{D}} \left[\, \theta'_d - \theta_{l_d\, r_d} \,\right]^+}_{\text{truck/drone waiting time}} + \underbrace{\lambda \sum_{d \in \mathcal{D}} \Big[\max\{\theta'_d,\, \theta_{l_d\, r_d}\} - D_d \Big]^+}_{\text{drone endurance penalty}},$$

where $[a]^+ = \max\{a, 0\}$.

The first term represents the total truck travel time along the route $\mathcal{C} = (c_1, \ldots, c_K)$. The second term accounts for the waiting time incurred when the truck reaches the recovery node before the drone. For each sortie d, θ'_d is the drone travel duration, while $\theta_{l_d\, r_d}$ is the truck travel time between the launch and recovery stages l_d and r_d. A positive difference therefore

corresponds to the truck waiting for the drone. The third term, inspired by the work of Gonzalez-R et al. (2020), penalizes violations of the drone endurance limit D_d. For each sortie, the maximum of the truck and drone travel durations (i.e., $\max\{\theta'_d, \theta_{l_d r_d}\}$) represents the operation time in which the drone is deployed. Whenever this exceeds the endurance limit, the excess time penalizes the objective by a coefficient λ.

In our ALNS, λ is dynamically adjusted during the search to regulate the balance between feasible and infeasible solutions, following the adaptive penalty adjustment principle used by Vidal et al. (2013). Let ρ_t denote the proportion of feasible candidate solutions observed during the most recent adjustment period, and let $\rho^\star$ be the target feasible proportion. At each adjustment step, the penalty coefficient is updated as $\lambda \leftarrow \eta^+\lambda$ if $\rho_t < \rho^\star$, and $\lambda \leftarrow \eta^-\lambda$ otherwise, where $\eta^+ > 1$ and $0 < \eta^- < 1$ are the increase and decrease factors.

4.2. Initial solution generation procedure

The initial solution generation procedure is straightforward and listed as follows. Since the FSTSP is a variant of the well-known TSP, any truck-only TSP tour constitutes a feasible solution. Given an instance with n customers, we first generate a random permutation of the customers. The depot is then added at the beginning and at the end of the sequence to complete the tour. Finally, the local search procedure presented in Section 4.4 is applied to improve the quality of the initial solution.

4.3. Destroy–repair group operators

In the FSTSP, destroy and repair operations on the truck route and drone sorties are strongly interdependent. Applying them independently may therefore create inconsistent intermediate solutions or make repair difficult. We address this issue by organizing destroy–repair pairs into *group operators*, where each group defines a coherent set of operators acting on a specific part of the solution structure. We use two groups: (i)truck-only modification and (ii) mixed truck–drone modification.

Type 1: Truck-only group. This group modifies the truck route $\mathcal{C}$, while keeping all launch and recovery references of existing drone sorties unchanged. It explores large neighborhoods of the truck tour without altering the temporal structure of drone operations. The **destroy operators** are: (i) *truck random removal*, which removes a randomly selected subset of truck customers; (ii) *truck worst removal*, which removes the customer at stage k with the largest arc contribution $\tau_{c_{k-1}c_k} + \tau_{c_k c_{k+1}}$; and (iii) *truck block removal*, which removes all customers in a consecutive block of stages. The **repair operators** are: (i) *truck greedy insertion*, which reinserts each removed customer at the least-cost position; (ii) *truck random insertion*, which uses randomly selected feasible positions; and (iii) *LDR 1/LDR 2* (Mara et al., 2022), which insert a customer at a stage minimizing $\tau_{c_{k-1}c_k}$ or $\tau_{c_k c_{k+1}}$.

Type 2: Mixed truck–drone group. This group simultaneously modifies the truck route and drone sorties, enabling broader structural changes than truck-only modifications. The **destroy operators** are: (i) *mixed block removal*, which removes truck customers in a selected route interval $[i, j]$ and any drone sortie whose interval $[l_d, r_d]$ intersects $[i, j]$; and (ii) *mixed outlier removal*, which removes a randomized subset of farthest customers from the depot, together with adjacent truck customers and any affected drone sorties. The **repair operators** are: (i) *mixed block repair*, which reconstructs the affected segment using nearest-neighbor truck insertion followed by greedy drone insertion; and (ii) *mixed outlier repair*, which reinserts removed customers in decreasing depot-distance order and decides whether each customer is better restored on the truck route or a drone sortie.

Together, these two groups provide a structured mechanism for diversification while avoiding conflicts between truck and drone decisions. Their combined effect enables large solution modifications before the local search.

4.4. Local search

The purpose of the local search phase is to intensify the exploration of the solution space by exhaustively examining a set of neighborhood structures specifically designed for the FSTSP. Six move types are considered: Add

Drone (AD), four variants of Swap Drone (SD1–SD4), and Remove Drone (RD). Local search follows a deterministic and cyclic sequence of moves:

$$\text{AD} \rightarrow \text{SD1} \rightarrow \text{SD2} \rightarrow \text{SD3} \rightarrow \text{SD4} \rightarrow \text{RD},$$

after which the sequence restarts from AD.

For each move in this sequence, the corresponding neighborhood is exhaustively explored in a best–first search manner. Notice that all candidate moves of the current type are evaluated in $O(1)$ (see Appendix A), and the move that provides the best improvement is immediately applied. The search with the current move continues until no further improvement from the same move can be found. At that point, the algorithm advances to the next move in the sequence. When the RD operator has been fully explored and no additional improvement is possible, one full cycle has been completed. The local search will be terminated after two full cycles.

Move definitions. Six neighborhood moves used during local search are presented below. Each move operates on the current solution structure and is illustrated in Figure 2.

1. *AD (Add Drone).* This move removes a customer from the truck route and creates a new drone sortie to serve that customer. The launch and recovery stages of the new sortie are selected from the truck route such that the new sortie does not overlap with any existing sortie. The launch and recovery stages of the existing sorties remain unchanged.
2. *SD1 (Swap two drone customers).* This move exchanges the drone-served customers of two distinct sorties while preserving the original launch and recovery stages of both sorties (and thus the truck route).
3. *SD2 (Shift a shared stage).* This move applies to two consecutive sorties such that the recovery stage of the first sortie coincides with the launch stage of the second sortie. It shifts this shared stage to another feasible stage on the truck route, thereby modifying the recovery stage of the first sortie and the launch stage of the second sortie.

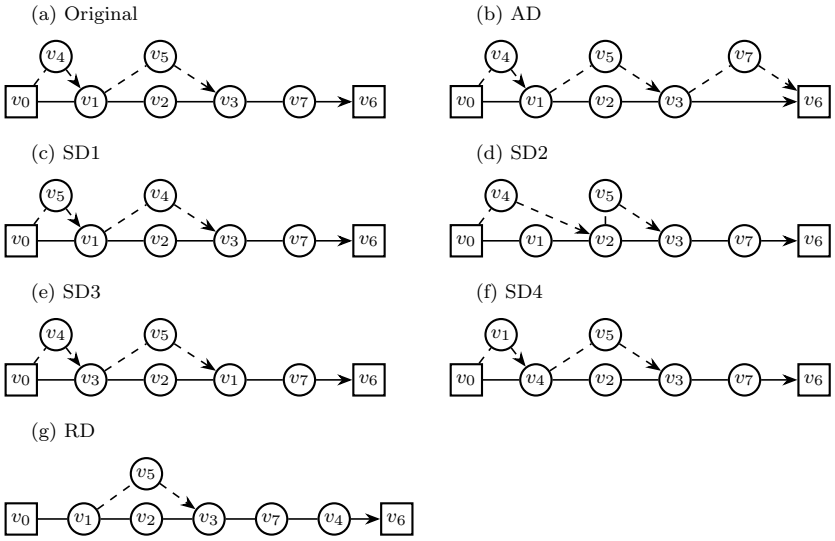

Figure 2: Illustration of the original solution and six local-search moves. Solid arcs denote truck movements and dashed arcs denote drone sorties: (a) Current solution, (b) AD creates a sortie for v_7; (c) SD1 swaps drone customers v_4 and v_5; (d) SD2 shifts the shared stage from v_1 to v_2; (e) SD3 swaps launch/recovery nodes v_1 and v_3; (f) SD4 swaps truck customer v_1 with drone customer v_4; and (g) RD reinserts drone customer v_4 into the truck route.

4. *SD3 (Swap launch and recovery truck customers).* This move swaps the truck customers located at the launch and recovery stages of the same sortie. As a result, the local truck-route structure is modified, and the corresponding sortie is updated accordingly.
5. *SD4 (Swap a truck customer and a drone customer).* This move swaps the drone-served customer of a sortie with either its launch-stage customer or its recovery-stage customer. Consequently, one customer changes from drone service to truck service, while the other changes from truck service to drone service.
6. *RD (Remove Drone).* This move removes an existing drone sortie and reinserts its drone-served customer into the truck route. The removed sortie is deleted, while others preserve their launch and recovery stages.

For each candidate move, we compute the objective variation in constant time, rather than recomputing the full value from scratch. The detailed

derivations for local-search moves are provided in Appendix A.

4.5. Speeding up Local Search with Static Move Descriptors

To reduce repeated evaluations in local search, we incorporate a lightweight Static Move Descriptor (SMD) mechanism, originally introduced by Zachariadis and Kiranoudis (2010) and later adapted in routing algorithms such as FILO (Accorsi and Vigo, 2024). The key idea is that a local-search move usually modifies only a small part of the solution; hence, most candidate moves remain unchanged after an accepted move and do not need to be re-evaluated (Beek et al., 2018).

In our ALNS, SMD is applied to the SD1 neighborhood, which is the most time-consuming local-search move. Each descriptor stores the two sorties involved in the swap and the corresponding precomputed objective variation Δ_{SD1}. At the beginning of the SD1 search, all candidate moves are evaluated once, and only improving moves are kept in an active list. When the best descriptor is applied, descriptors that conflict with it, i.e., those involving affected sorties, are removed and re-evaluated. Unaffected descriptors are retained without recomputation.

This implementation uses a simple linear container rather than specialized data structures. Although this does not change the worst-case complexity, it substantially reduces unnecessary move evaluations in practice while preserving the simplicity of the proposed local-search framework.

5. Computational Experiments and Analysis

To guide the computational study, we formulate seven research questions covering the main aspects of the proposed ALNS which aim to evaluate its competitiveness, the contribution of key algorithmic components, the effect of drone speed, and the robustness of the algorithm across independent runs.

- *RQ1. How competitive is the proposed ALNS in solution quality and runtime?* We compare ALNS with the state-of-the-art algorithms MD–SPP–H (Schaumann et al., 2025), HGA–TAC (Mahmoudinazlou and Kwon, 2024), EP–All (Agatz et al., 2018), and SPP–All (Kundu

et al., 2022) to assess its solution quality and computational time across different customer distributions and instance sizes.

- *RQ2. What is the contribution of each algorithmic component?* We investigate the marginal contribution of each component of the proposed algorithm through an ablation study, focusing on the local search phase and different operator groups.

- *RQ3. How effective is the Static Move Descriptor mechanism?* We evaluate SMD by comparing local search time, and overall ALNS runtime, with and without this mechanism.

- *RQ4. Why is the two-cycle local search configuration adopted?* We explain the choice of the two-cycle setting by comparing different local search cycle configurations in terms of solution quality and runtime.

- *RQ5. What is the impact of the dynamic penalty mechanism?* We evaluate whether allowing controlled exploration of infeasible solutions improves the search performance compared with a strict feasible-only strategy.

- *RQ6. How does drone speed affect the cost-saving performance of the proposed ALNS?* Using equal truck and drone speeds as the baseline, we analyzes how the benefit obtained by ALNS changes when the drone becomes faster relative to the truck.

- *RQ7. How robust is the proposed ALNS with respect to different independent runs?* We assess the stability and robustness by comparing repeated-run objective values with the best value found for each instance and reporting the proportions within several tolerance levels.

We use the 234 benchmark instances of Agatz et al. (2018), which are widely adopted in the truck–drone routing literature. These instances contain 50 to 500 customers under three spatial distributions: uniform, single-center, and double-center. For benchmark comparisons, we use the 150 instances with 50, 75, 100, 175, and 250 customers, with 10 instances for

each size–distribution combination. The remaining 84 instances with 375 and 500 customers are reported as additional references for future studies. Travel times are based on Euclidean distances, drone endurance is set to 20, 40, or infinity, and launch/recovery service times are omitted, following Agatz et al. (2018). All experiments were conducted on Ubuntu 24.10 with a 13th Gen Intel(R) Core(TM) i5-13500 CPU and 62 GB RAM, using a single CPU core. The algorithm was implemented in C++ and built using CMake 3.30.3. ALNS and dynamic-penalty parameters are reported in Appendix B. The complete computational results are reported in the Supplementary Material.

Since ALNS, MD–SPP–H, and HGA–TAC are stochastic, each was run independently 10 times per instance. We record the best and average objective values over the 10 runs, denoted by z^{best} and $\bar{z}$, respectively. For each benchmark method Z, the best- and average-objective gaps of ALNS are defined as

$$\Delta_Z^{\text{best}} = 100 \times \frac{z_{\text{ALNS}}^{\text{best}} - z_Z^{\text{best}}}{z_Z^{\text{best}}}(\%), \qquad \Delta_Z^{\text{avg}} = 100 \times \frac{\bar{z}_{\text{ALNS}} - \bar{z}_Z}{\bar{z}_Z}(\%).$$

For each distribution–size group $\mathcal{I}$, we report the corresponding averages

$$\bar{\Delta}_Z^{\text{best}} = \frac{1}{|\mathcal{I}|} \sum_{\mathcal{I}} \Delta_Z^{\text{best}}, \qquad \bar{\Delta}_Z^{\text{avg}} = \frac{1}{|\mathcal{I}|} \sum_{\mathcal{I}} \Delta_Z^{\text{avg}}.$$

When the benchmark method or variant is clear from the column group, the tables report these values simply as $\bar{\Delta}^{\text{best}}$ and $\bar{\Delta}^{\text{avg}}$.

To assess robustness across ALNS runs, let z_{ir} be the objective value from run r on instance i, and let $z_i^{\text{best}} = \min_r z_{ir}$ be the best value found over the 10 runs. We define

$$\Delta_{ir} = 100 \times \frac{z_{ir} - z_i^{\text{best}}}{z_i^{\text{best}}}.$$

For a tolerance level ϵ, we report

$$\rho(\epsilon) = \frac{1}{|\mathcal{I}|R} \left| \{(i, r) : \Delta_{ir} \leq \epsilon\} \right|,$$

where $R = 10$ is the number of runs per instance. Thus, $\rho(1\%)$, $\rho(2\%)$, and $\rho(5\%)$ are the fractions of runs within 1%, 2%, and 5% of the best value found for the corresponding instance.

5.1. Comparative performance against benchmark algorithms - RQ1

To answer the RQ1, we report comparative results for unlimited drone endurance ($DTL = \infty$), finite endurance ($DTL = 20$ and $DTL = 40$). We also report the computation results for large-scale instances with 375 and 500 customers.

5.1.1. Results under unlimited drone endurance ($DTL = \infty$)

Tables 1 and 2 compare ALNS with MD–SPP–H, HGA–TAC, EP–All, and SPP–All in solution quality and runtime under $DTL = \infty$.

Table 1: Average gap comparison between the proposed ALNS and benchmark algorithms by customer distribution and instance size.

Distribution	n	MD-SPP-H		HGA-TAC		EP-All		SPP-All	
		$\overline{\Delta}^{\text{best}}$	$\overline{\Delta}^{\text{avg}}$	$\overline{\Delta}^{\text{best}}$	$\overline{\Delta}^{\text{avg}}$	$\overline{\Delta}^{\text{best}}$	$\overline{\Delta}^{\text{avg}}$	$\overline{\Delta}^{\text{best}}$	$\overline{\Delta}^{\text{avg}}$
Uniform	50	-5.78	-6.09	-7.78	-8.71	-10.04	-9.34	-9.77	-9.07
Uniform	75	-3.46	-3.73	-7.72	-8.17	-9.07	-7.97	-8.56	-7.46
Uniform	100	-3.69	-4.83	-8.19	-8.87	-10.08	-9.28	-9.34	-8.54
Uniform	175	-2.63	-3.35	-6.49	-6.66	-7.31	-6.10	-6.62	-5.39
Uniform	250	-2.78	-2.91	-6.18	-6.05	-5.32	-4.07	-4.95	-3.68
1-center	50	-1.66	-2.36	-5.33	-6.47	-6.34	-5.70	-4.95	-4.29
1-center	75	-1.76	-3.06	-7.20	-8.08	-7.77	-6.91	-5.75	-4.87
1-center	100	-2.25	-3.32	-7.40	-8.51	-7.19	-6.37	-6.74	-5.89
1-center	175	-2.86	-3.72	-7.70	-8.11	-4.98	-3.82	-3.81	-2.62
1-center	250	-2.69	-2.88	-6.15	-6.19	-4.04	-2.98	-3.24	-2.16
2-center	50	-4.32	-4.85	-7.22	-8.00	-8.76	-8.24	-8.65	-8.12
2-center	75	-4.49	-4.88	-7.97	-9.17	-8.87	-7.69	-8.68	-7.52
2-center	100	-4.76	-5.39	-8.98	-9.54	-8.29	-7.20	-7.42	-6.36
2-center	175	-3.34	-2.79	-9.17	-7.87	-6.91	-4.18	-6.51	-3.74
2-center	250	-3.79	-2.37	-8.68	-7.18	-5.43	-2.86	-4.51	-1.91

Note: Gap values are percentages. Negative values indicate that ALNS obtains a lower objective value than the corresponding benchmark algorithm.

ALNS consistently achieves better solution quality across all distribution–size groups and against all benchmark algorithms, both in best and average objective value. Over all tested groups, the average best-objective gaps are

−3.35%, −7.48%, −7.36%, and −6.63% against MD–SPP–H, HGA–TAC, EP–All, and SPP–All, respectively; the corresponding average-objective gaps are −3.77%, −7.84%, −6.18%, and −5.44%. The gains are especially pronounced relative to HGA–TAC, EP–All, and SPP–All.

Table 2: Average running time comparison between the proposed ALNS and benchmark algorithms by customer distribution and instance size.

Distribution	n	ALNS	MD-SPP-H	HGA-TAC	EP-All	SPP-All
Uniform	50	7.45	4.81	5.00	2.29	0.04
Uniform	75	16.24	24.18	9.24	11.37	0.23
Uniform	100	26.49	71.50	13.42	52.75	1.28
Uniform	175	77.23	489.14	38.94	623.45	11.37
Uniform	250	179.49	1397.54	73.80	3158.70	43.39
1-center	50	8.48	7.00	5.39	2.32	0.07
1-center	75	18.39	32.82	10.25	12.27	0.23
1-center	100	31.15	102.60	15.86	56.97	1.15
1-center	175	95.52	662.78	36.28	809.47	8.21
1-center	250	201.13	1528.56	66.59	2979.01	50.13
2-center	50	8.49	6.96	4.74	2.72	0.44
2-center	75	16.91	31.53	8.92	13.79	0.72
2-center	100	30.31	87.17	13.79	42.94	2.11
2-center	175	90.40	714.70	26.83	602.90	20.50
2-center	250	209.59	1558.63	36.10	2905.77	95.31

Note: Runtime values are average computational times in seconds over the instances in each distribution–size group.

In runtime, HGA–TAC and SPP–All are generally faster than ALNS, but they deliver worse solution quality across all tested groups. In contrast, ALNS becomes substantially faster than MD–SPP–H and EP–All on medium and large instances; for $n \geq 100$, it is faster than both methods in every distribution. On 250-customer instances, MD–SPP–H requires about 1,300 seconds and EP–All nearly 3,000 seconds, whereas ALNS remains below 210 seconds. Overall, ALNS offers the best quality–runtime trade-off among the compared methods.

5.1.2. Results under finite drone endurance ($DTL = 20$ and $DTL = 40$)

We next evaluate ALNS under finite drone endurance, $DTL = 20$ and $DTL = 40$. Figure 3 compare ALNS with MD–SPP–H and HGA–TAC adapted to these settings.

The proposed ALNS handles endurance-constrained truck–drone synchronization much more effectively than HGA–TAC. Across almost all tested

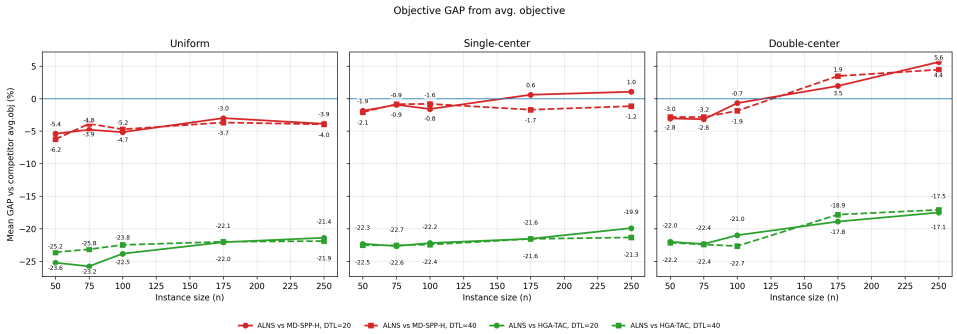

Figure 3: Comparison with external baselines across three instance distributions: uniform, single-center, and double-center.

groups, ALNS consistently outperforms HGA–TAC by a very large margin. For $DTL = 20$, the average best-objective gaps against HGA–TAC range from -20% to -25% across all distributions and instance sizes. Similarly, for $DTL = 40$, the average best-objective gaps remain around -19% to -24%. The corresponding average-objective gaps show a similar pattern.

Compared with MD–SPP–H, the improvement is smaller but still generally positive. For the uniform instances, ALNS consistently achieves negative gaps under both $DTL = 20$ and $DTL = 40$, with average best-objective improvements typically between -3% and -6%. The gains are smaller on the single-center instances and become mixed for some large double-center instances. For example, on the 2-center instances with $n = 250$, the average best-objective gaps become slightly positive. This suggests that these tightly clustered spatial structures combined with finite drone endurance create particularly difficult synchronization patterns for the search process.

Overall, ALNS remains highly competitive under finite-endurance constraints.

5.1.3. Results on large-scale instances ($n = 375$ and $n = 500$)

Table 3 reports ALNS performance on large-scale instances with $n = 375$ and $n = 500$. The average gap $\bar{\rho}$ remains small at 1.81% for $n = 375$ and 1.76% for $n = 500$, indicating stable performance across repeated runs.

These results suggest that ALNS scales well to instances with a large

Table 3: Summary of ALNS performance on large-scale instances with $n = 375$ and $n = 500$.

n	Distribution	#Inst.	Time (s)	z^{best}	$\bar{z}$	$\bar{\rho}$ (%)
375	1-center	14	603.2	1603.95	1624.59	1.29
375	2-center	14	596.0	2202.32	2266.74	2.92
375	Uniform	15	517.9	961.48	973.46	1.25
375	Overall	43	571.1	1574.65	1606.53	1.81
500	1-center	10	1522.1	1891.52	1918.65	1.43
500	2-center	10	1395.3	2624.00	2688.21	2.46
500	Uniform	11	1318.4	1111.46	1127.35	1.43
500	Overall	31	1408.9	1851.01	1886.11	1.76

Note: z^{best} and $\bar{z}$ denote the best and average objective values over repeated ALNS runs, respectively. The value $\bar{\rho}$ is the average relative deviation between $\bar{z}$ and z^{best} within each instance group; it measures run-to-run stability rather than a comparison gap against another algorithm.

number of nodes. Although runtime increases with instance size, the small gap between the best and average objective values indicates limited sensitivity to randomization and reliable performance on large-scale truck–drone routing problems.

5.2. Ablation analysis of algorithmic components - RQ2

Table 4 addresses RQ2 by comparing the full ALNS with three reduced variants: without local search, without the truck-only operator group, and without the mixed truck–drone operator group. The gaps are computed relative to the full ALNS, so a positive value indicates worse solution quality after removing the corresponding component.

The results show a clear hierarchy among the algorithmic components. Local search is the dominant contributor: removing it increases the best objective value by 24.61% on average and the average objective value by 27.63%. This confirms that local search is the main intensification mechanism of ALNS, especially on large instances.

The truck-only operator group provides the next largest contribution. Removing it increases the best and average objective values by 3.54% and 4.21%, respectively, with a consistent effect across almost all distribution–

Table 4: Ablation study: comparison between the full ALNS and variants with removed components

Distribution	n	w/o Local Search		w/o Truck-only Group		w/o Mixed Group	
		$\overline{\Delta}^{\text{best}}$	$\overline{\Delta}^{\text{avg}}$	$\overline{\Delta}^{\text{best}}$	$\overline{\Delta}^{\text{avg}}$	$\overline{\Delta}^{\text{best}}$	$\overline{\Delta}^{\text{avg}}$
Uniform	50	15.67	18.92	2.87	3.88	1.01	2.03
Uniform	75	19.93	25.00	3.81	4.02	1.72	2.35
Uniform	100	29.17	32.47	3.98	5.04	1.55	2.63
Uniform	175	36.69	38.10	3.77	4.35	1.80	2.69
Uniform	250	39.11	39.38	4.22	5.19	1.30	1.79
1-center	50	10.88	13.63	3.36	4.51	1.03	1.78
1-center	75	16.75	19.78	2.58	2.96	1.29	1.89
1-center	100	21.60	24.13	2.76	3.68	0.96	1.76
1-center	175	28.42	31.62	3.35	4.06	0.69	0.69
1-center	250	33.50	34.53	2.80	4.17	0.43	1.03
2-center	50	13.28	16.46	3.53	4.64	0.65	1.54
2-center	75	14.55	19.79	3.19	3.48	0.63	1.55
2-center	100	22.28	29.33	4.06	4.96	1.11	2.25
2-center	175	32.41	34.52	4.50	3.78	1.62	2.34
2-center	250	34.88	36.86	4.38	4.42	1.62	2.30
Overall	–	24.61	27.63	3.54	4.21	1.16	1.91

Note: Gap values are percentages and are computed relative to the full ALNS. A positive value indicates that removing the corresponding component worsens the solution quality.

size groups. This indicates that truck-only modifications remain important for reaching high-quality solutions.

The mixed truck–drone operator group has a smaller but still positive effect. Removing it worsens the best and average objective values by 1.16% and 1.91%, respectively, indicating a useful additional contribution to diversification and coordination. So overall, the ablation study supports keeping all three components in the final ALNS framework.

5.3. Impact of Static Move Descriptors on runtime - RQ3

Figure 4 addresses RQ3 by evaluating the computational contribution of the Static Move Descriptor mechanism at two levels. Speedup is defined as the runtime without SMD divided by the runtime with SMD, so values above one indicate a reduction in computational time.

Figure 4(a) reports the speedup for the SD1 local-search operator alone. We compare move-evaluation time with and without SMD. SMD already yields a speedup of about 2 on 50-customer instances, increasing to roughly

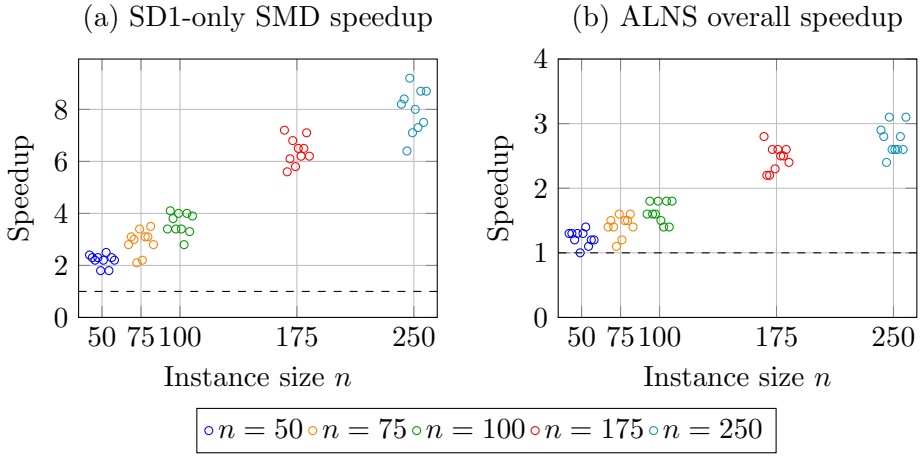

Figure 4: Speedup impact of SMD in SD1-only and ALNS overall with instance size n

6–8 on larger instances. The increase is approximately linear with instance size, which is clearly visible in Figure 4(a). This indicates that SMD is especially effective when repeated move evaluations become more expensive.

Figure 4(b) reports the overall speedup of the complete ALNS algorithm. Although the effect is smaller than for SD1 alone, the speedup remains above one for all tested instance sizes, showing that the local acceleration from SMD translates into a meaningful reduction in total ALNS runtime.

5.4. Why is the two-cycle local search configuration adopted - RQ4

Table 5 addresses RQ4 by comparing one-cycle, two-cycle, and three-cycle local search configurations, using the two-cycle setting as the reference.

The two-cycle configuration consistently outperforms the one-cycle configuration in both best and average objective value. This indicates that a single local search cycle is insufficient to fully exploit the neighborhoods, whereas a second cycle provides useful additional refinement. The improvement is moderate but systematic across all customer distributions and instance sizes.

The comparison between two-cycle and three-cycle configurations is mixed. The three-cycle configuration occasionally obtains slightly better solutions, but the differences are small and inconsistent, while runtime increases no-

Table 5: Comparison of one-cycle, two-cycle, and three-cycle local search configurations

Distribution	n	2-cycle vs. 1-cycle		2-cycle vs. 3-cycle		Average runtime (s)		
		$\overline{\Delta}^{\text{best}}$	$\overline{\Delta}^{\text{avg}}$	$\overline{\Delta}^{\text{best}}$	$\overline{\Delta}^{\text{avg}}$	1-cycle	2-cycle	3-cycle
Uniform	50	-0.29	-0.34	0.16	0.13	5.08	7.45	9.55
Uniform	75	-0.80	-0.49	-0.08	0.09	10.78	16.24	20.29
Uniform	100	-0.14	-0.49	-0.06	-0.05	17.82	26.49	34.15
Uniform	175	-0.38	-0.54	0.35	0.23	54.13	77.23	103.63
Uniform	250	-0.50	-0.52	-0.03	0.14	119.23	179.49	223.88
1-center	50	-0.36	-0.27	-0.06	0.12	5.69	8.48	10.91
1-center	75	-0.52	-0.60	0.08	0.04	12.16	18.39	22.41
1-center	100	-0.52	-0.86	-0.05	-0.02	20.15	31.15	39.06
1-center	175	-0.44	-0.51	0.34	0.34	63.59	95.52	117.87
1-center	250	-0.62	-0.81	0.74	0.28	136.82	201.13	243.70
2-center	50	-0.28	-0.52	0.10	0.09	5.36	8.49	10.21
2-center	75	-0.27	-0.46	0.15	0.01	10.75	16.91	21.75
2-center	100	-0.74	-1.64	-0.22	-0.53	19.44	30.31	36.67
2-center	175	-0.50	-0.90	0.03	0.36	60.01	90.40	112.07
2-center	250	-0.41	-1.29	0.06	-0.09	142.26	209.59	247.16

Note: Gap values are percentages. A negative gap means that the two-cycle configuration obtains a lower objective value than the compared configuration. The two-cycle configuration is used as the default setting.

ticeably, especially on large instances. Thus, the potential benefits of a third cycle do not justify its additional computational cost.

Overall, the two-cycle configuration provides the best balance between solution quality and runtime and is therefore used as the default setting.

5.5. Impact of the dynamic penalty mechanism - RQ5

To assess the contribution of the dynamic penalty mechanism, we compare the proposed `ALNS_dynamic_penalty` with `ALNS_bigM_penalty`, which uses a fixed large penalty value. This mechanism is relevant only for finite drone endurance settings, $DTL \in \{20, 40\}$, since no endurance violation can occur when $DTL = \infty$.

These results show that the dynamic penalty mechanism is an important part of the proposed ALNS framework. Compared with a fixed big-M penalty, it enables controlled exploration of promising infeasible solutions early in the search while progressively steering the search toward feasibility. This is especially valuable when endurance and synchronization constraints interact strongly.

Table 6: Average performance gaps of `ALNS_dynamic_penalty` relative to `ALNS_bigM_penalty` in percentages. Negative values indicate improvements.

Instance class	DTL	#Inst.	$\overline{\Delta}^{\text{best}}$	$\overline{\Delta}^{\text{avg}}$
Single-center	20	50	−5.00	−7.92
Single-center	40	50	−7.50	−12.34
Double-center	20	50	−14.30	−25.36
Double-center	40	50	−22.12	−30.88
Uniform	20	50	−28.60	−37.84
Uniform	40	50	−7.46	−23.00
Overall	–	300	−14.16	−22.89

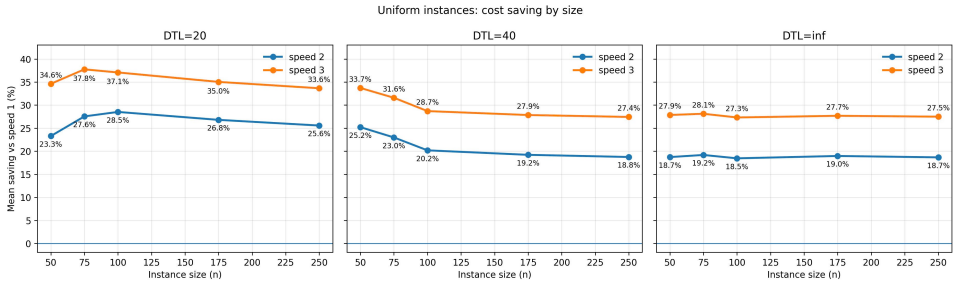

Figure 5: Impact of drone speed on cost savings.

5.6. Impact of drone speed - RQ6

The relative speed of the drone compared to that of the truck is an important factor influencing the achievable cost savings. Using the case of drone speed ratio equal to 1 as the baseline, Figure 5 reports the mean cost savings achieved when the drone speed is increased to twice and three time the truck speed. For $DTL = 20$, increasing the drone speed to a ratio of 2 yields cost savings ranging from 23.3% to 28.5%, while a speed ratio of 3 further increases to between 33.6% and 37.8%. For $DTL = 40$, we notice that the savings decrease gradually as the instance size increases. The savings range from from 25.2% to 18.8% for speed ratio 2, and from 33.7% to 27.4% for speed ratio 3.

When $DTL = \infty$, the cost savings become more stable across different instance sizes. In this setting, the speed ratio of 2 provides savings of approximately 18.5%–19.2%, whereas the speed ratio of 3 achieves savings of

approximately 27.3%–28.1%. These results show that increasing the drone speed consistently improves the cost-saving performance. However, the additional benefit from increasing the speed ratio from 2 to 3 is moderate. This suggests diminishing marginal returns from drone speed improvement, which is likely due to the synchronization requirement. Once the drone becomes sufficiently fast, it may have to wait more frequently for the truck, limiting the additional savings obtained from further speed increases.

5.7. Robustness of the algorithm with respect to different runs - RQ7

Table 7 reports the dispersion of ALNS across 224 benchmark instances and 2240 runs under $DTL = \infty$, including instance sizes up to $n = 500$. Overall, 48.44% of the runs are within 1% of the best value found for their instance, 78.35% are within 2%, 90.85% are within 3%. The results indicate that ALNS remains stable across independent runs, with 96.03% of all runs falling within 5% of the per-instance best value.

The robustness pattern differs across spatial distributions. The single-center instances are the most stable, reaching 86.49% within 2% and 100.0% within 5%. Uniform instances also show strong stability, with 96.45% of runs within 3% and 99.87% within 5%. The double-center instances are more dispersed, with a maximum observed gap of 15.32%. Nevertheless, even for this most difficult distribution, 97.97% of the runs remain within 10% of the best solution found. The results confirm the robustness and stable performance of the proposed algorithm across independent runs.

Table 7: Dispersion of ALNS across independent runs under unlimited drone endurance

Distribution	$\rho(1\%)$	$\rho(2\%)$	$\rho(3\%)$	$\rho(5\%)$	$\rho(10\%)$	$\Delta^{\max}$ (%)
Uniform	47.11	79.74	96.45	99.87	100.00	5.66
1-center	52.16	86.49	97.57	100.00	100.00	4.50
2-center	46.08	68.78	78.38	88.11	97.97	15.32
Overall	48.44	78.35	90.85	96.03	99.33	15.32

Note: Each value gives the percentage of ALNS runs whose objective value is within the corresponding tolerance of the best value found by ALNS for the same instance over 10 independent runs. $\Delta^{\max}$ denotes the maximum gap to that per-instance best value.

6. Conclusion

This paper propose an Adaptive Large Neighborhood Search Algorithm to solve the Flying Sidekick Traveling Salesman Problem. Compared to the state-of-art algorithms, our ALNS consistently achieves better solution quality at the aggregated level, both in terms of the best solution obtained over repeated runs and in terms of the average solution quality across all distribution–size groups. We notice that the magnitude of the improvement varies across customer distributions, depending on the benchmark algorithms. Overall, the proposed framework is robust across different spatial structures and is not limited to a specific distribution.

Regarding computational time, our proposed solution achieves a favorable trade-off between solution quality and computational time. While some benchmark algorithms are faster, their speed comes at the expense of noticeably worse solution quality. Conversely, the methods that are closer in solution quality, require much longer computational times on large instances. Thus, our ALNS provides a competitive and scalable alternative for FSTSP.

We further test our ALNS solution on multiple large-scale instances. Overall, the algorithm remains stable on these large instances, with small gaps between the average best objective value and the average objective value. These small gaps indicate that the algorithm is robust across repeated runs, even when the number of customers increases to 500.

We also perform ablation analysis to clarify the algorithmic contributions of different components of our ALNS framework and further validate the benefits of SMD integration and the dynamic penality mechanism in our solution. In the end, our proposed ALNS provides an effective tool to address FSTSP. We also believe that this framework can be successfully extended for other challenging problems, such as TSP-D with multiple drops.

References

Accorsi, L., Vigo, D., 2024. Routing one million customers in a handful of minutes. Computers & Operations Research 164, 106562. doi:10.1016/j.cor.2024.106562.

Agatz, N., Bouman, P., Schmidt, M., 2018. Optimization approaches for the traveling salesman problem with drone. Transportation Science 52, 965–981. doi:10.1287/trsc.2017.0791.

Beek, O., Raa, B., Dullaert, W., Vigo, D., 2018. An efficient implementation of a static move descriptor-based local search heuristic. Computers & Operations Research 94, 1–10. doi:10.1016/j.cor.2018.01.006.

Boccia, M., Mancuso, A., Masone, A., Sterle, C., 2023. A new milp formulation for the flying sidekick traveling salesman problem. Networks 82, 254–276. doi:10.1002/net.22172.

Boccia, M., Masone, A., Sforza, A., Sterle, C., 2021. A column-and-row generation approach for the flying sidekick travelling salesman problem. Transportation Research Part C: Emerging Technologies 124, 102913. doi:10.1016/j.trc.2020.102913.

Bouman, P., Agatz, N., Schmidt, M., 2018. Dynamic programming approaches for the traveling salesman problem with drone. Networks 72, 528–542. doi:10.1002/net.21864.

Boysen, N., Fedtke, S., Schwerdfeger, S., 2020. Last-mile delivery concepts: a survey from an operational research perspective. OR Spectrum 43, 1–58. doi:10.1007/s00291-020-00607-8.

De Freitas, J.C., Penna, P.H.V., 2020. A variable neighborhood search for flying sidekick traveling salesman problem. International Transactions in Operational Research 27, 267–290. doi:10.1111/itor.12671.

Gonzalez-R, P.L., Canca, D., Andrade-Pineda, J.L., Calle, M., Leon-Blanco, J.M., 2020. Truck-drone team logistics: A heuristic approach to multi-drop route planning. Transportation Research Part C: Emerging Technologies 114, 657–680. doi:10.1016/j.trc.2020.02.030.

Ha, Q.M., Deville, Y., Pham, Q.D., Hà, M.H., 2018. On the min-cost traveling salesman problem with drone. Transportation Research Part C: Emerging Technologies 86, 597–621. doi:10.1016/j.trc.2017.11.015.

Kundu, A., Escobar, R.G., Matis, T.I., 2022. An efficient routing heuristic for a drone-assisted delivery problem. IMA Journal of Management Mathematics 33, 583–601. doi:10.1093/imaman/dpab039.

Li, H., Chen, J., Wang, F., Bai, M., 2021. Ground-vehicle and unmanned-aerial-vehicle routing problems from two-echelon scheme perspective: A review. European Journal of Operational Research 294, 1078–1095. doi:10.1016/j.ejor.2021.02.022.

Macrina, G., Di Puglia Pugliese, L., Guerriero, F., Laporte, G., 2020. Drone-aided routing: A literature review. Transportation Research Part C: Emerging Technologies 120, 102762. doi:10.1016/j.trc.2020.102762.

Mahmoudinazlou, S., Kwon, C., 2024. A hybrid genetic algorithm with type-aware chromosomes for traveling salesman problems with drone. European Journal of Operational Research 318, 719–739. doi:10.1016/j.ejor.2024.05.009.

Mara, S.T.W., Rifai, A.P., Sopha, B.M., 2022. An adaptive large neighborhood search heuristic for the flying sidekick traveling salesman problem with multiple drops. Expert Systems with Applications 205, 117647. doi:10.1016/j.eswa.2022.117647.

Moradi, N., Wang, C., Mafakheri, F., 2024. Urban air mobility for last-mile transportation: A review. Vehicles 6, 1383–1414. doi:10.3390/vehicles6030066.

Moshref-Javadi, M., Winkenbach, M., 2021. Applications and research avenues for drone-based models in logistics: A classification and review. Expert Systems with Applications 177, 114854. doi:10.1016/j.eswa.2021.114854.

Murray, C.C., Chu, A.G., 2015. The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery. Transportation Research Part C: Emerging Technologies 54, 86–109. doi:10.1016/j.trc.2015.03.005.

Nguyen, T.C., Nguyen, H.D., Le, H.T., Kaneko, S., 2022. Residents' preferred measures and willingness-to-pay for improving urban air quality: A case study of Hanoi city, Vietnam. Journal of Economics and Development 24, 262–275. doi:10.1108/JED-03-2021-0036.

Poikonen, S., Golden, B., Wasil, E.A., 2019. A branch-and-bound approach to the traveling salesman problem with a drone. INFORMS Journal on Computing 31, 335–346. doi:10.1287/ijoc.2018.0826.

Roberti, R., Ruthmair, M., 2021. Exact methods for the traveling salesman problem with drone. Transportation Science 55, 315–335. doi:10.1287/trsc.2020.1017.

Ropke, S., Pisinger, D., 2006. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation science 40, 455–472. doi:10.1287/trsc.1050.0135.

Schaumann, S.K., Kundu, A., Pina-Pardo, J.C., Winkenbach, M., Gatica, R.A., Wagner, S.M., Matis, T.I., 2025. The flying sidekick traveling salesman problem with multiple drops: An effective heuristic approach. Computers & Industrial Engineering 210, 111523. doi:10.1016/j.cie.2025.111523.

Vidal, T., Crainic, T.G., Gendreau, M., Prins, C., 2013. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. Computers & Operations Research 40, 475–489. doi:https://doi.org/10.1016/j.cor.2012.07.018.

Zachariadis, E.E., Kiranoudis, C.T., 2010. A strategy for reducing the computational complexity of local search-based methods for the vehicle routing problem. Computers & Operations Research 37, 2089–2105. doi:10.1016/j.cor.2010.02.009.

Appendix A. Move Evaluation

Each neighborhood operator modifies either the truck route, the set of drone sorties, or both. To support efficient neighborhood exploration, we

evaluate the objective variation of every candidate move in $O(1)$ time after an $O(n)$ preprocessing step on the current solution.

More precisely, once a solution is fixed, we precompute: (i) the truck travel time between any two truck stages, so that θ_{ij} can be queried in constant time for any pair of stages $i < j$; (ii) the drone travel time θ'_s of each sortie s; and (iii) the constant-time references needed to identify the few sorties that may be affected by a local modification, such as the sortie covering a given truck stage or the sorties immediately adjacent to a given sortie. Therefore, each move only changes a constant number of truck arcs and a constant number of sortie-delay terms.

For convenience, we use the following local truck-route primitives throughout this section. Removing the customer at truck stage k induces the change

$$\delta_k^{\mathrm{rm}} = \tau_{c_{k-1}c_{k+1}} - \tau_{c_{k-1}c_k} - \tau_{c_k c_{k+1}}.$$

Inserting a customer x between truck stages $i-1$ and i induces the change

$$\delta_i^{\mathrm{ins}}(x) = \tau_{c_{i-1}x} + \tau_{xc_i} - \tau_{c_{i-1}c_i}.$$

When needed, a direct truck-route change can be written as a combination of one removal term and one insertion term, in that order.

Definition 1 (delay function). Let $\theta_{l_s r_s}$ and θ'_s be the truck and drone travel durations, respectively, between the launch stage l_s and the recovery stage r_s of a sortie s. The delay contribution of sortie s is

$$\mathcal{D}(\theta_{l_s r_s}, \theta'_s) = [\theta'_s - \theta_{l_s r_s}]^+ + \lambda[\max\{\theta_{l_s r_s}, \theta'_s\} - D_d]^+. \tag{A.1}$$

Proposition 1 (change in delay under perturbations). Let $\Delta t, \Delta t' \in \mathbb{R}$ be the changes in truck and drone travel durations of a sortie s. The corresponding

change in its delay contribution is

$$\begin{aligned}\omega(s,\Delta t,\Delta t') &= \mathcal{D}(\theta_{l_s r_s}+\Delta t,\theta'_s+\Delta t')-\mathcal{D}(\theta_{l_s r_s},\theta'_s).\\ &= [\theta'_s+\Delta t'-(\theta_{l_s r_s}+\Delta t)]^+-[\theta'_s-\theta_{l_s r_s}]^+ \\ +\lambda\Big([\max\{\theta'_s+\Delta t',\theta_{l_s r_s}+\Delta t\}-D_d]^+&-[\max\{\theta'_s,\theta_{l_s r_s}\}-D_d]^+\Big).\end{aligned} \tag{A.2}$$

For convenience, let $\mathrm{cover}(k)$ denote the unique sortie whose launch–recovery interval contains truck stage k, when such a sortie exists. Similarly, let $\mathrm{prev}(s)$ and $\mathrm{next}(s)$ denote the sorties immediately before and after s that share its launch or recovery stage, respectively, when such sorties exist.

We now apply Proposition 1 to the six neighborhood moves.

AD move. This move removes the truck customer at stage k and creates a new drone sortie with launch stage k_1 and recovery stage k_2.
Objective change. $\Delta_{\mathrm{AD}} = \delta_k^{\mathrm{rm}}+A_k^{\mathrm{AD}}+\mathcal{D}\Big(\theta_{k_1k_2}+B_{k_1,k,k_2}^{\mathrm{AD}},\tau'_{c_{k_1}c_k}+\tau'_{c_kc_{k_2}}\Big)$, where

$$A_k^{\mathrm{AD}} = \begin{cases}\omega(\mathrm{cover}(k),\delta_k^{\mathrm{rm}},0), & \text{if } \mathrm{cover}(k) \text{ exists},\\ 0, & \text{otherwise},\end{cases}$$

$$B_{k_1,k,k_2}^{\mathrm{AD}} = \begin{cases}\delta_k^{\mathrm{rm}}, & \text{if } k_1<k<k_2,\\ 0, & \text{otherwise.}\end{cases}$$

Interpretation. The term δ_k^{rm} is the truck shortcut after removing c_k. If stage k lies inside another sortie interval, its truck duration is updated by A_k^{AD}. The last term is the delay of the new sortie, with truck duration adjusted by $B_{k_1,k,k_2}^{\mathrm{AD}}$ when $k_1<k<k_2$. Hence Δ_{AD} is evaluated in $O(1)$.

SD1 move. The SD1 operator exchanges the drone-served customers of two distinct sorties u and v while keeping their launch and recovery stages unchanged. Let $u=(c_{u1},c_{u2},c_{u3})$, $v=(c_{v1},c_{v2},c_{v3})$.

Objective change. Define

$$\Delta'_u = \tau'_{c_{u1}c_{v2}} + \tau'_{c_{v2}c_{u3}} - \tau'_{c_{u1}c_{u2}} - \tau'_{c_{u2}c_{u3}},$$
$$\Delta'_v = \tau'_{c_{v1}c_{u2}} + \tau'_{c_{u2}c_{v3}} - \tau'_{c_{v1}c_{v2}} - \tau'_{c_{v2}c_{v3}}.$$

Then $\Delta_{\mathrm{SD1}} = \omega(u, 0, \Delta'_u) + \omega(v, 0, \Delta'_v)$.
Interpretation. SD1 leaves the truck route unchanged and only updates the drone durations of sorties u and v. Hence Δ_{SD1} is the sum of two delay updates and is computed in $O(1)$ time.

SD2 move. The SD2 move applies to two consecutive sorties u and v with $v = u + 1$ and $r_u = l_v$. The move shifts their shared truck stage to a new feasible stage k, with $l_u < k < r_v$. Before the move, $u = (c_{u1}, c_{u2}, c_{u3})$, $v = (c_{v1}, c_{v2}, c_{v3})$, where $c_{u3} = c_{v1}$. After the move, the updated sorties are $u' = (c_{u1}, c_{u2}, c_k)$, $v' = (c_k, c_{v2}, c_{v3})$. The truck- and drone-duration changes are $\Delta t_u = \theta_{l_u k} - \theta_{l_u r_u}; \Delta'_u = \tau'_{c_{u2}c_k} - \tau'_{c_{u2}c_{u3}}, \Delta t_v = \theta_{k r_v} - \theta_{l_v r_v}, \Delta'_v = \tau'_{c_k c_{v2}} - \tau'_{c_{v1}c_{v2}}$.
Objective change. $\Delta_{\mathrm{SD2}} = \omega(u, \Delta t_u, \Delta'_u) + \omega(v, \Delta t_v, \Delta'_v)$.
Interpretation. SD2 changes the shared truck stage between two consecutive sorties. Each sortie may therefore change its truck duration and one drone arc. No other part of the solution is affected, so the move is evaluated in $O(1)$ time.

SD3 move. The SD3 move swaps the launch-stage customer and the recovery-stage customer of the same sortie s. Let $l = l_s$ and $r = r_s$, and let the original sortie be $s = (c_l, c_{s2}, c_r)$. After the move, the sortie becomes $s' = (c_r, c_{s2}, c_l)$.
Objective change. The objective variation is decomposed into four terms:

$$\Delta_{\mathrm{SD3}} = \Delta^{\mathrm{SD3}}_{\mathrm{trk}} + \omega\big(s, \Delta t^{\mathrm{SD3}}_s, \Delta'^{,\mathrm{SD3}}_s\big) + A^{\mathrm{SD3}}_{\mathrm{prev}} + A^{\mathrm{SD3}}_{\mathrm{next}}.$$

The direct truck-route contribution is

$$\Delta^{\mathrm{SD3}}_{\mathrm{trk}} = \begin{cases} \delta^{\mathrm{rm}}_r + \delta^{\mathrm{ins}}_l(c_r), & \text{if } r - l = 1, \\ \delta^{\mathrm{rm}}_r + \delta^{\mathrm{ins}}_l(c_r) + \delta^{\mathrm{ins}}_r(c_l) - \tau_{c_{r-1}c_{r+1}}, & \text{if } r - l > 1. \end{cases}$$

The truck- and drone-duration changes of sortie s are

$$\Delta t_s^{\mathrm{SD3}} = \begin{cases} \tau_{c_r c_l} - \tau_{c_l c_r}, & \text{if } r - l = 1, \\ \tau_{c_r c_{l+1}} + \tau_{c_{r-1} c_l} - \tau_{c_l c_{l+1}} - \tau_{c_{r-1} c_r}, & \text{if } r - l > 1, \end{cases}$$

$$\Delta_s'^{,\mathrm{SD3}} = \tau'_{c_r c_{s2}} + \tau'_{c_{s2} c_l} - \tau'_{c_l c_{s2}} - \tau'_{c_{s2} c_r}.$$

The additional adjustment for a previous neighboring sortie is

$$A_{\mathrm{prev}}^{\mathrm{SD3}} = \begin{cases} \omega\Big(\mathrm{prev}(s), \tau_{c_{l-1} c_r} - \tau_{c_{l-1} c_l}, \tau'_{c_{p2} c_r} - \tau'_{c_{p2} c_l}\Big), & \text{if prev}(s) \text{ exists}, \\ 0, & \text{otherwise}, \end{cases}$$

where c_{p2} denotes the drone-served customer of $\mathrm{prev}(s)$. The additional adjustment for a next neighboring sortie is

$$A_{\mathrm{next}}^{\mathrm{SD3}} = \begin{cases} \omega\big(\mathrm{next}(s), \tau_{c_l c_{r+1}} - \tau_{c_r c_{r+1}}, \tau'_{c_l c_{n2}} - \tau'_{c_r c_{n2}}\big), & \text{if next}(s) \text{ exists}, \\ 0, & \text{otherwise}, \end{cases}$$

where c_{n2} denotes the drone-served customer of $\mathrm{next}(s)$.

Interpretation. SD3 removes c_r from stage r and inserts it at stage l; when $r - l > 1$, customer c_l is then inserted at stage r. The same swap also changes the truck and drone durations of sortie s. If the launch or recovery stage is shared with a neighboring sortie, that sortie is updated as well. Hence the move is evaluated in $O(1)$ time.

SD4 move. The SD4 move swaps the drone-served customer of a sortie with either its launch-stage customer or its recovery-stage customer. We present the launch-stage case; the recovery-stage case follows symmetrically. Let s be the considered sortie, with launch stage $l = l_s$ and recovery stage $r = r_s$. Before the move, $s = (c_l, c_{s2}, c_r)$, and after swapping c_l with c_{s2}, the updated sortie becomes $s' = (c_{s2}, c_l, c_r)$.

Objective change. The objective variation of the launch-stage case is

$$\Delta_{\mathrm{SD4,L}} = \Delta_{\mathrm{trk}}^{\mathrm{SD4,L}} + \omega\big(s, \Delta t_s^{\mathrm{SD4,L}}, \Delta_s'^{,\mathrm{SD4,L}}\big) + A_{\mathrm{prev}}^{\mathrm{SD4,L}}.$$

The direct truck-route contribution is $\Delta_{\mathrm{trk}}^{\mathrm{SD4,L}} = \delta_l^{\mathrm{rm}} + \delta_l^{\mathrm{ins}}(c_{s2})$. The truck- and drone-duration changes of sortie s are

$$\Delta t_s^{\mathrm{SD4,L}} = \tau_{c_{s2}c_{l+1}} - \tau_{c_l c_{l+1}},$$
$$\Delta_s'^{,\mathrm{SD4,L}} = \tau'_{c_{s2}c_l} + \tau'_{c_l c_r} - \tau'_{c_l c_{s2}} - \tau'_{c_{s2}c_r}.$$

The additional correction for a previous neighboring sortie is

$$A_{\mathrm{prev}}^{\mathrm{SD4,L}} = \begin{cases} \omega\Big(\mathrm{prev}(s), \tau_{c_{l-1}c_{s2}} - \tau_{c_{l-1}c_l}, \tau'_{c_{p2}c_{s2}} - \tau'_{c_{p2}c_l}\Big), & \text{if prev}(s) \text{ exists,} \\ 0, & \text{otherwise,} \end{cases}$$

where c_{p2} denotes the drone-served customer of prev(s).

Interpretation. SD4 removes c_l from stage l and inserts c_{s2} at the same stage. It also changes the truck and drone durations of sortie s, and it may affect one previous neighboring sortie if that sortie recovers at stage l. Hence the move is evaluated in $O(1)$ time.

RD move. The RD move removes an existing drone sortie s and reinserts its drone-served customer into the truck route. Let $s = (c_{s1}, c_{s2}, c_{s3})$, where c_{s2} is the customer currently served by the drone. Suppose that c_{s2} is inserted between truck stages $k-1$ and k.

Objective change. The objective variation is $\Delta_{\mathrm{RD}} = -\mathcal{D}(\theta_{l_s r_s}, \theta'_s) + \delta_k^{\mathrm{ins}}(c_{s2}) + A_k^{\mathrm{RD}}$, where

$$A_k^{\mathrm{RD}} = \begin{cases} \omega(\mathrm{cover}(k), \delta_k^{\mathrm{ins}}(c_{s2}), 0), & \text{if cover}(k) \text{ exists and cover}(k) \neq s, \\ 0, & \text{otherwise.} \end{cases}$$

Interpretation. The first term removes the delay contribution of sortie s; the second inserts c_{s2} back into the truck route. If the insertion position lies inside another sortie interval, its truck duration is updated by A_k^{RD}. Hence Δ_{RD} is evaluated in $O(1)$ time.

Appendix B. Algorithm and Penalty Parameter Settings

The ALNS parameters are listed in Table B.8 and Table B.9. Settings in Table B.8 largely follow Mara et al. (2022), with minor adjustments to the final temperature and destroy ratio.

Table B.8: ALNS parameter configuration

Parameter	Description	Value
T_1	Initial temperature	200
T_0	Final temperature	10
α	Cooling rate	0.99999
BZ	Boltzmann constant	0.1
$\Omega_1, \Omega_2, \Omega_3, \Omega_4$	Weight rewards	0.13, 0.15, 0, 0
λ	Initial penalty coefficient	1
λ'	Decay parameter	0.9
f	Destroy ratio	Random in $[0.25, 0.75]$

Table B.9 reports the initial ranges and the calibrated values (tuned by `irace`) used in the final experiments for the dynamic penalty mechanism: update interval, target proportion of feasible candidate solutions, penalty increase factor, and penalty decrease factor. The tuning budget was set to 10,000 algorithm runs on a single machine. The training set was derived from the benchmark instances used in the main experiments. For each instance, four additional perturbed instances were generated. In each perturbed instance, every customer coordinate (x, y) was independently multiplied by a random factor sampled uniformly from $[0.9, 1.1]$. Only the perturbed instances were shuffled before being passed to `irace`. During tuning, the algorithm seed was fixed to isolate the effect of the penalty parameters from stochastic variation in the search.

Table B.9: Dynamic penalty parameters calibrated by `irace`

Parameter	Description	Initial range	Selected value
I_λ	Penalty update interval	$(10, 500)$	300
$\rho^\star$	Target feasible proportion	$(0.05, 1.00)$	0.1
η^+	Penalty increase factor	$(1.05, 2.00)$	1.2
η^-	Penalty decrease factor	$(0.50, 0.95)$	0.8