

MODEMS: A Tailored Solution for Promoting On-Campus Passenger Mobility

Mostafa Emam

`mostafa.emam@unibw.de`

Bundeswehr University Munich

Björn Martens

Bundeswehr University Munich

Thomas Rottmann

Bundeswehr University Munich

Matthias Gerds Gerds

Bundeswehr University Munich

Research Article

Keywords: Electric Vehicle Routing Problem, Pickup and Delivery, Time Windows, VRPPDTW, DARP, MILP, On-Demand Mobility

Posted Date: October 13th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-8769914/v2>

License:   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

MODEMS: A Tailored Solution for Promoting On-Campus Passenger Mobility

Mostafa Emam^{1*}, Björn Martens¹, Thomas Rottmann¹,
Matthias Gerdts¹

^{1*}Institute of Applied Mathematics and Scientific Computing, University
of the Bundeswehr Munich, Werner-Heisenberg-Weg 39, Neubiberg,
85579, Bavaria, Germany.

*Corresponding author(s). E-mail(s): mostafa.emam@unibw.de;
Contributing authors: bjoern.martens@unibw.de;
thomas.rottmann@unibw.de; matthias.gerdts@unibw.de;

Abstract

MODEMS (Munich **O**n-**D**emand **E**lectric **M**obility **S**ervice) is a simulation-based dispatching and routing framework for on-campus ridepooling; it augments existing shared micromobility by autonomously transporting passengers between fixed service stations using a small heterogeneous fleet of Electric Vehicles (EVs). We formulate the problem as a variant of the Electric Autonomous Dial-A-Ride Problem (e-ADARP) with heterogeneous agent capacities and load-dependent energy consumption, for which we develop three Mixed-Integer Linear Programming (MILP) models with different treatments of Time Window (TW) violation, request rejection, and State-of-Charge (SoC) calculation. Furthermore, we implement an Adaptive Large Neighborhood Search (ALNS) to ensure the generation of feasible solutions under strict time requirements, while constructive heuristic methods provide feasible warm starts for the MILPs to accelerate convergence. Requests can be registered beforehand or emerge dynamically during operation, with varying demand, service duration, and desired pickup times. The dynamic routing elements are managed using a simulation-based rolling-horizon framework, which periodically constructs a static routing instance with a restricted set of relevant requests, updated agent states, and current network data. Finally, we evaluate our developed strategies through numerical experiments on a digital representation of the campus network, discuss the achieved results, and present potential future research directions.

Keywords: Electric Autonomous Dial-A-Ride Problem, Dynamic Vehicle Routing, Ridepooling, Heterogeneous Fleet, MILP, ALNS, On-Demand Campus Mobility

1 Introduction

Shared mobility services enhance accessibility within large campuses by complementing walking, cycling, and private motorized vehicles. At the University of the Bundeswehr Munich, the *MORE Sharing* project offers several micromobility solutions, including e-bikes and e-scooters, and empirical studies have demonstrated the efficacy of these services in promoting on-campus mobility [1, 2]. However, conventional micromobility inherently necessitates the ability to operate vehicles, which may be inadequate for individuals with limited mobility. Therefore, we propose a complementary ride-pooling service, utilizing electric autonomous shuttle buses, to facilitate passenger transportation between designated service stations located on the university grounds.

Vehicle dispatching and routing fall under the broader category of Vehicle Routing Problems (VRPs), where one or more vehicles are deployed to complete transportation requests, subject to operational constraints [3]. Relevant extensions include hard or soft time windows, simultaneous pickup and delivery operations, mixed or heterogeneous vehicles, dynamic requests, and energy-related considerations [4–6]. These elements are independently or jointly modeled in specialized variants, such as the Pickup and Delivery Problem with Time Windows (PDPTW) and the Electric VRP with Time Windows (EVRPTW) [3, 7]. A noteworthy variant that focuses on passenger transportation is the Dial-A-Ride Problem (DARP), which emphasizes service-quality metrics, such as maximum passenger ride times and individualized scheduling [5, 8]. An advanced variant thereof is the Electric Autonomous Dial-A-Ride Problem (e-ADARP), which involves electric autonomous vehicles with battery restrictions [9–11].

MODEMS offers a tailored e-ADARP variant operating a limited fleet of heterogeneous electric shuttles; the vehicles differ in maximum capacities and initial and minimum State-of-Charge (SoC), while agent-specific notation accommodates different load-free and load-dependent energy consumption parameters. Owing to the current campus infrastructure, charging operations may span several hours, hindering the integration of charging decisions in the routing problem. Instead, we develop an independent, deterministic operational workflow, guiding agents with a SoC below a specified threshold to return to a depot, where agents are charged and become available after a delay with an updated (partial or full) SoC. Furthermore, we impose minimum SoC constraints in the routing models to guarantee that agents reserve sufficient energy to return to a depot after trip completion.

To address the dynamic nature of requests and agent states, we establish a deterministic rolling-horizon approach that composes a new static routing problem every ten minutes or whenever a request is received with an earliest pickup within the planning horizon. This simplifies the modeling process and limits the routing problem to a relevant subset of requests, enabling the application of exact methods under strict time requirements [12–14]. Moreover, the models distinguish between previously accepted and newly received requests, ensuring the continuity of decision-making across horizons and, simultaneously, supporting request selectivity.

Since we expect small problem instances, exact methods with Mixed-Integer Linear Programming (MILP) formulations represent a natural choice; they offer optimality guarantees for the computed solutions, contingent upon sufficient computational time and/or resources. Nevertheless, given the NP-hardness of our problem, the solution

time is inherently unpredictable and increases rapidly with problem size and complexity, rendering it unreliable for a dynamic mobility service [3, 15, 16]. This can be mitigated by warm-starting the MILP solver with a feasible initial solution, acting as the starting point and initial upper bound [17]. Alternatively, we may utilize heuristics and metaheuristics, which have been extensively researched for VRP variants. For instance, Adaptive Large Neighborhood Search (ALNS) has been successfully applied to PDPTW, DARP, EVRPTW, and dynamic multi-depot routing problems [12, 18–20].

1.1 Scope

This paper investigates the routing and scheduling layer of MODEMS and its placement within a conceptual service architecture. The routing problem comprises heterogeneous autonomous shuttles, dynamically emerging requests, fixed physical service stations, free-depot selection, soft or hard TWs, selective request acceptance, pickup and delivery precedence constraints, capacity limits, load-dependent energy consumption, request-specific service duration, and maximum passenger ride time. Charging decisions are decoupled from the routing model and are independently undertaken by a deterministic operational workflow. Furthermore, we support closed and open routing objectives [21], both of which include energy guarantees to avoid mid-route battery depletion. In this work, we implement the routing strategies within a rolling-horizon framework, which we numerically evaluate using deterministic routing instances on a simulated campus network. The remaining service architecture layers, namely the frontend, backend, database, information service, communication middleware, and agent coordination, are presented as a conceptual Proof-of-Concept (PoC), while their software implementation and real-world integration are deferred to future work.

1.2 Contribution

Our contribution is twofold: first, we present an application-oriented service architecture designed to address the complexities of electric autonomous on-campus ridepooling. It integrates the unique combination of fixed service stations, a small heterogeneous fleet, dynamically emerging and rejectable requests, load-dependent discharging, and user-satisfaction metrics. Second, we conduct a comparative study of three MILP formulations and an ALNS approach, utilizing aligned objective functions and comparable operational modes. MILP1 mandates the acceptance of all received requests, permits violations of the TW constraints, and employs a simplified energy consumption model based on conservative driving time estimation. MILP2 provides request selectivity, hard TWs, and load-dependent SoC determination. MILP3 combines both models, offering selective service, soft TWs, and load-dependent SoC propagation. ALNS complements the exact methods through a variety of standard and problem-specific destroy and repair operators, selected to balance intensification and diversification [22]. In addition, ALNS employs feasibility-driven repair and destroy operations, thereby accelerating the search process. Subsequently, this work constitutes a comparative analysis, integration, and evaluation of complementary methods, specifically tailored to address our application-driven mobility problem.

In the next section, we discuss relevant research that has informed our approach, after which we define our problem components in Section 3. We explain the MILP models and ALNS strategy in Section 4, followed by the conceptual MODEMS architecture in Section 5, including the complete workflow and management of dynamic elements. Section 6 describes the conducted simulation experiments and achieved results; finally, we present our concluding remarks and ideas for future work in Section 7.

2 Literature Review

We begin with survey papers on VRP [3, 6] and DARP [5, 8, 23], which provide a comprehensive overview of common problem variants, mathematical formulations, objectives, and solution methods. Foundational concepts of the mathematical modeling of mixed-logic systems can be found in [24, 25]. Traditional DARP models employ two-index or three-index MILP formulations to represent agent routes, pickup and delivery dependencies, passenger loads, TWs, and maximum ride times [5, 8]. Although these models are intelligible and readily accommodate application-specific requirements, their solution spaces grow combinatorially with the number of visited locations, potentially impeding the solution process for larger problem instances. In contrast, event-based formulations encode service events and current passenger occupancy in the graph nodes, implicitly satisfying capacity, pairing, precedence, and same-agent requirements [11, 16, 26]. Despite outperforming location-based formulations on standard DARP instances, event-based formulations remain relatively novel and have not yet been sufficiently studied in the literature. Furthermore, problem setup times explode with capacities larger than conventional taxis, rendering the approach unattractive for shuttle buses [11].

Owing to the increasing demand for EVs, the EVRPTW was introduced as a MILP in [7], alongside a hybrid algorithm combining Variable Neighborhood Search (VNS) and Tabu Search (TS) for tackling large instances. Herein, linear relationships were employed to model the charging and load-independent discharging processes. Subsequent studies include advanced operational elements, such as time-dependent waiting at recharging stations [20], load-dependent energy consumption and nonlinear charging [27], and battery-swapping decisions for dynamic routing of electric taxis [28]. A comprehensive review of EVRP variants and algorithms is available at [6]. In general, these contributions highlight the significance of integrating discharging and charging decisions within the routing of EVs; however, high-fidelity models often introduce additional variables and constraints, further complicating the solution process.

Dynamic routing problems are characterized by information that emerges or changes during operation, often mandating continuous route updates or repeated re-optimization [4, 8, 14]. The problem can be modeled as a Markov Decision Process and solved using approximate dynamic programming, or, more commonly, a snapshot is deterministically converted into a static instance and solved using exact or heuristic methods. For instance, a multi-depot dynamic VRPTW is studied in [12], in which the routing problem is periodically compiled using current agent locations, remaining capacities, and a request list, and subsequently solved with ALNS. Furthermore, a dynamic DARP is formulated as an event-based graph and efficiently solved using a rolling-horizon algorithm in [13],

where request classification preserves currently active routing decisions and enables re-optimization of scheduled requests. Both approaches achieve short response times for the investigated instances. Nevertheless, they remain predominantly reactive and deterministic, without anticipating stochastic requests or uncertain travel conditions.

Shared Autonomous Electric Vehicles (SAEVs) extend the routing problem beyond automation by coupling request assignment, vehicle re-balancing, and parking and charging decisions [29]. Here, a well-studied passenger-oriented variant is the e-ADARP, for which static and dynamic frameworks exist with a Branch-and-Cut (B&C) method for the static problem and a LNS for online operation [30]. Recent exact strategies include a Branch-and-Price (B&P) method utilizing battery-restricted passenger fragments [10], a bi-objective framework independently minimizing agent travel time and excess passenger ride time [9], and event-based MILPs with strengthened temporal and SoC constraints [11]. These methods substantially improve the solution of e-ADARP instances and emphasize the multi-objective compromise between operational efficiency, customer satisfaction, and SoC considerations. However, their applicability to more complex operational environments remains inhibited by the combinatorial growth of route fragments and event graphs, in addition to simplifying assumptions, such as static requests, linear charging, and load-independent energy consumption.

In terms of solution strategies, exact solution frameworks, such as B&C and B&P, can provide certificates of optimality when convergence is achieved [10, 30]. Herein, feasible warm starts may accelerate convergence by providing an initial incumbent solution and upper bound [17]. In contrast, heuristics and metaheuristics typically begin with a feasible initial solution and iteratively improve it until a stopping criterion is met. A particularly interesting metaheuristic is ALNS, which was originally developed to address PDPTWs by effectively exploring large neighborhoods through the adaptive application of destroy and repair operators [18]. Since then, it has been successfully adapted to a wide variety of DARP applications, including constant-time feasibility-driven insertions [19], customer-oriented ride and waiting-time minimization [31], and flexible TWs and delivery locations [32]. A recent review and ranking of ALNS operators identifies the most promising destroy and repair strategies, emphasizing that operator effectiveness remains problem-dependent [22].

To conclude, existing literature covers a diverse range of theoretical and practical techniques to address DARP and ridepooling problems, including formulations, energy-aware modeling, dynamic component management, adoption of autonomous agents, and efficient solution strategies. Nevertheless, we have not identified a study that jointly handles our unique combination: fixed service stations, a small heterogeneous fleet of autonomous EVs, dynamically emerging and rejectable requests with TWs, limited charging capabilities, free-depot selection, and short response times. We address this gap through an application-oriented service architecture, wherein dynamic elements are resolved using a rolling-horizon approach to deterministically compile a snapshot as a static routing problem. Subsequently, the problem is solved using three MILP formulations and an ALNS, for which we conduct a comparative performance analysis under strict time requirements. Our contribution lies in the integration and evaluation of these complementary approaches for our practically-motivated mobility service, in addition to our feasibility-driven request management that expedites the search process of ALNS.

3 Problem Components

In this section, we present the three core elements of our problem and their mathematical formulation: the road network, user requests, and fleet agents.

3.1 Road Network

First, we distinguish physical and virtual road networks. Given that our approach is tailored for the university grounds, existing physical depots act as hubs for parking and charging of fleet agents. Moreover, we may exploit historical data to determine the locations of service stations for user pickup and drop-off, cf. Fig. 1. Consequently, we model the physical network as a directed graph $G = (V, E)$, where nodes $V = H \cup S$ comprise the physical hubs and stations, and edges E represent their corresponding travel arcs. Each arc is associated with a navigational route specifying the shortest-path distance and road speed limits, which enables the determination of a lower bound on arc travel time $\underline{c}_{ij} > 0, \forall i, j \in V, i \neq j$, and $\underline{c}_{ii} = 0$. Since $\underline{c}_{ij}$ is computed with the implicit assumptions of driving at the speed limit without adverse traffic, it only constitutes the minimum physically feasible travel time. Furthermore, we integrate a variability factor $\mu \in [1.1, 1.25]$ to determine the actual travel time $c_{ij} = \mu \underline{c}_{ij} > 0, \forall i, j \in V, i \neq j$, accounting for route uncertainty and reasonable delays. μ remains constant on the network-level to facilitate exploiting triangle inequality properties. Alternatively, routing engines, e.g., GraphHopper [33], can be utilized for dynamic route optimization, incorporating real-time traffic data and yielding more accurate travel times.

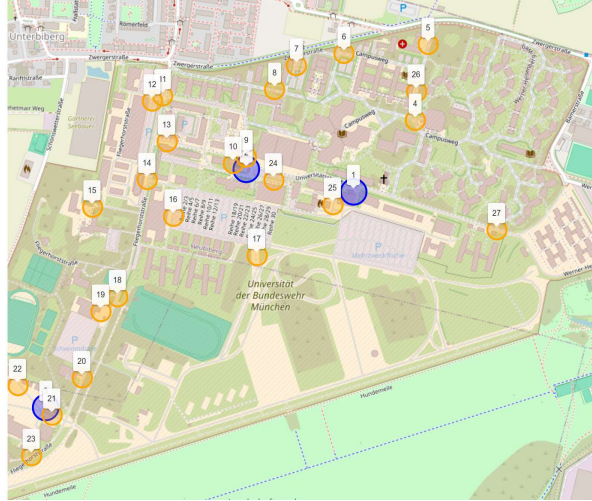


Fig. 1 Example configuration for the campus road network using OpenStreetMap (OSM)[34]

Next, we introduce the virtual road network; it is derived from the physical network and offers a more concise representation for the MILP models. For each agent $k \in K$, we create a virtual initial node v^k , mapped to its current or anticipated physical location in V , which can be a hub or station. The set of agents' starting nodes is defined as

$H_0 = \bigcup_{k \in K} \{v^k\}$. For each request $r \in R$, we create virtual pickup p^r and delivery d^r nodes, each mapped to its corresponding physical station in S , such that different requests retain unique virtual nodes even when they share a physical pickup or delivery station. Accordingly, we define the sets $S^p = \bigcup_{r \in R} \{p^r\}$ and $S^d = \bigcup_{r \in R} \{d^r\}$, with the combined set of virtual service stations $S' = S^p \cup S^d$. Lastly, we have the set of virtual depots H_f containing copies of the physical hubs, where agents may terminate their trips. The combination of these mutually disjoint sets yields the complete set of virtual-network nodes $V' = H_0 \cup S^p \cup S^d \cup H_f$ and arcs $E' = V' \times V'$, whose travel times c_{ij} are inferred by mapping each virtual node to its corresponding location in the physical road network. Infeasible arc types are explicitly excluded in the MILP formulations.

Since routing algorithms strictly use the virtual road network $G' = (V', E')$, we consistently employ the terms *hub*, *station*, *node*, *network* in the upcoming sections to denote *virtual* elements, unless otherwise stated. Additionally, we interchangeably use the terms *initial nodes* and *initial depots* for H_0 , *service stations* and *stations* for S' , *pickup nodes* and *pickup stations* for S^p , *delivery nodes* and *delivery stations* for S^d , and *final nodes* and *final depots* for H_f .

3.2 User Requests

Receiving, processing, and archiving requests fall under the service architecture, which is discussed in Section 5. From a routing perspective, requests are classified into two disjoint categories: (i) newly added requests R_n , which the routing strategy may accept or reject, and (ii) scheduled requests R_s , which were accepted in a previous planning cycle, have not yet begun service, and must remain accepted. This yields the set of requests $R = R_n \cup R_s$ considered in the current routing epoch. Each request $r \in R$ is defined as a tuple $(p^r, d^r, q^r, s^r, [e^r, l^r])$, where p^r, d^r are the pickup and delivery nodes, respectively, $q^r \in \mathbb{Z}^+$ is the passenger demand, and $s^r \geq 0$ is the required service time for both pickup and drop-off operations. Herein, users only specify the desired earliest start of service $e^r > 0$ at pickup p^r , while the latest service start is computed using the fixed TW $\omega > 0$, with $l^r = e^r + \omega$. The request status and routing problem variant jointly determine the possibility of TW violations, as explained in Table 1. Note that a hard maximum ride time is enforced for both soft TW and hard TW variants to improve user satisfaction.

Table 1 Pickup TW restrictions based on request status and problem variant

Request	Problem	Early Service $t < e^r$	Late Service $t > l^r$
New	Soft TW	permitted up to ω , penalized	permitted but penalized
	Hard TW	omitted, must wait until $t = e^r$	prohibited
Scheduled	Soft TW	omitted, must wait until $t = e^r$	permitted but penalized
	Hard TW	omitted, must wait until $t = e^r$	prohibited

For consistency with existing literature, we duplicate request parameters at the virtual nodes, with $e_{p^r} = e^r$, $l_{p^r} = l^r$, and $s_i = s^r$, for $i \in \{p^r, d^r\}$. Similarly, $q_{p^r} = q^r$ and $q_{d^r} = -q^r$ represent embarking and disembarking of passengers, respectively.

3.3 Fleet Agents

We now establish the energy consumption model for our EVs. The university campus is predominantly flat with a negligible road gradient and is divided into two zones with known speed limits and road coefficients. Disregarding acceleration and deceleration, and assuming agents always drive at the speed limit, the only remaining variable is the gross vehicle weight, which is a function of the load-free vehicle mass and the current passenger load, cf. [27, 28]. With a constant vehicle mass, we introduce the conservative linear energy consumption model:

$$\mathcal{E} = \Phi_1 + \Phi_2 q, \quad (1)$$

which is independently valid for each of our two driving zones. Φ_1 is a constant accounting for rolling resistance, aerodynamic drag, and energy consumed by accessories in a load-free EV. Assuming an average passenger weight, Φ_2 accommodates the additional consumption due to the number of on-board passengers $q \in \mathbb{Z}_0^+$. By normalizing energy with respect to battery capacity, we can model the SoC drop per minute of driving as $\alpha + \beta q$, where α is the default discharge rate without passengers and β is the load-dependent discharge rate. In addition, we adopt the common proposal that SoC remains constant when an agent is stationary, e.g., when waiting at a service station [27, 28, 30].

Our fleet comprises three differently configured e-Crafters¹, each accommodating a different number of passengers; we conducted a number of test drives on the university grounds to empirically determine the energy-consumption parameters. In the first zone with a speed limit of 7 km h⁻¹, we have $\alpha = 6.65 \times 10^{-3} \text{ min}^{-1}$ and $\beta = 0.15 \times 10^{-3} \text{ min}^{-1}$. In the second zone with a speed limit of 30 km h⁻¹, we have $\alpha = 6.25 \times 10^{-3} \text{ min}^{-1}$ and $\beta = 0.15 \times 10^{-3} \text{ min}^{-1}$. Due to their similar construction, the differences in parameters across agents were negligible. To simplify the modeling process, we henceforth assume a worst-case discharge for all routes spanning one or both of these zones, with $\alpha = 6.65 \times 10^{-3} \text{ min}^{-1}$ and $\beta = 0.15 \times 10^{-3} \text{ min}^{-1}$; despite currently having the same discharge rates for all agents, we henceforth retain agent-specific notation to readily integrate different agents in the future.

We propose two representations for the fleet agents $k \in K$. In the extended tuple $(v^k, \delta^k, Q^k, \underline{\sigma}^k, \sigma^k, \alpha^k, \beta^k)$, $v^k \in H_0$ is the virtual starting node, $\delta^k \geq 0$ is the time of earliest availability at v^k , and $Q^k \in \mathbb{Z}^+$ denotes the maximum passenger capacity or load. The combination of v^k, δ^k facilitates the consideration of agents that may not be momentarily available in the routing process. Specifically, these parameters enable us to anticipate availability once certain conditions are met, such as after completing a delivery at a station or charging at a depot; this is explained in detail in Section 5. At the start of the workday, v^k is typically mapped to a physical depot in H and $\delta^k = 0$. For the energy-related parameters, we have the minimum operational SoC $\underline{\sigma}^k \in (0, 1]$ to safeguard against mid-course battery depletion, the initial SoC $\sigma^k \in [\underline{\sigma}^k, 1]$ at v^k , where $\sigma^k = 1$ signifies a fully charged battery, the default discharge rate $\alpha^k > 0$, and the load-dependent discharge rate $\beta^k > 0$. Furthermore, we have the reduced tuple $(v^k, \delta^k, Q^k, D^k)$, where $D^k \geq 0$ represents the remaining driving duration. Here, D^k imitates the SoC constraints by assuming a worst-case battery discharge per minute,

¹<https://www.volkswagen-vans.co.uk/en/about-us/van-life/ecrafter.html>

where:

$$D^k = \frac{\sigma^k - \underline{\sigma}^k}{\alpha^k + \beta^k Q^k} \quad (2)$$

4 Solution Methods

This section explains the MILP models and ALNS algorithm. In the sequel, we apply the Big-M method to linearize some timing constraints, wherein $M \in \mathbb{R}^+$ is derived from the rolling-horizon planning schedule to improve solver efficiency [28]. In addition, we accommodate an Open VRP (OVRP) variant, where we exclude travel to the final depots from the operational objective [21]. This is a practical design choice, as agents are allowed to park at any designated stop on campus grounds and do not need to return to a depot. Nevertheless, we maintain the return-to-depot arcs in the energy feasibility conditions for both closed and open variants, ensuring that each agent possesses sufficient energy to reach a freely selected depot following its final delivery.

4.1 MILP1: Soft Time Windows with Rudimentary SoC

In the first strategy, we present a non-selective variant that uses slack variables to permit TW violations and the basic agent tuple $(v^k, \delta^k, Q^k, D^k)$. We have the variables:

- $x_{ij}^k = \begin{cases} 1, & \text{if agent } k \in K \text{ travels from node } i \text{ to node } j, \text{ with } i, j \in V' \text{ and } i \neq j, \\ 0, & \text{otherwise.} \end{cases}$
- $t_i \geq 0$: start of service (arrival) time at node $i \in S'$
- $\tau_i \geq 0$: slack variable for the pickup TW at node $i \in S^p$; 0 if the TW is not violated
- $T \geq 0$: mission duration (maximum route-completion time across all agents)
- $z_i \in \mathbb{Z}_0^+$: passenger load upon departure from node $i \in S'$

Here, we employ the maximum route-completion time T , in lieu of the conventional summation of all agent travel times, to achieve a more even distribution of service across deployed agents. Additionally, we define two penalty parameters: a relatively small $\epsilon > 0$ to minimize request service-start times, and a comparatively larger $\zeta > \epsilon > 0$ to penalize TW violations. This yields the objective function:

$$\min \quad T + \sum_{r \in R} (\epsilon (t_{p^r} + t_{d^r}) + \zeta \tau_{p^r}) \quad (3)$$

which minimizes the mission duration, request service-start times, and pickup TW violations. Next, we delineate the request handling and flow conservation constraints.

$$\sum_{k \in K} \sum_{i \in (H_0 \cup S') \setminus \{p^r\}} x_{ip^r}^k = 1, \quad \forall r \in R \quad (4)$$

$$\sum_{i \in (H_0 \cup S') \setminus \{p^r\}} x_{ip^r}^k = \sum_{i \in S' \setminus \{d^r\}} x_{id^r}^k, \quad \forall r \in R, k \in K \quad (5)$$

$$\sum_{i \in (H_0 \cup S') \setminus \{p^r\}} x_{ip^r}^k = \sum_{j \in S' \setminus \{p^r\}} x_{p^r j}^k, \quad \forall r \in R, k \in K \quad (6)$$

$$\sum_{i \in S' \setminus \{d^r\}} x_{id^r}^k = \sum_{j \in (S' \cup H_f) \setminus \{d^r\}} x_{d^r j}^k, \quad \forall r \in R, k \in K \quad (7)$$

$$\begin{aligned} \sum_{i \in V'} x_{ii}^k + \sum_{i \in H_0 \setminus \{v^k\}} \sum_{j \in V'} x_{ij}^k + \sum_{j \in H_0 \cup S^d \cup H_f} x_{v^k j}^k \\ + \sum_{i \in S^p} \sum_{j \in H_0 \cup H_f} x_{ij}^k + \sum_{i \in S^d} \sum_{j \in H_0} x_{ij}^k + \sum_{i \in H_f} \sum_{j \in V'} x_{ij}^k = 0, \quad \forall k \in K \end{aligned} \quad (8)$$

$$\sum_{r \in R} x_{v^k p^r}^k \leq 1, \quad \forall k \in K \quad (9)$$

$$\sum_{i \in S'} \sum_{j \in S' \setminus \{i\}} x_{ij}^k \leq (|S'| - 1) \sum_{r \in R} x_{v^k p^r}^k, \quad \forall k \in K \quad (10)$$

Equation (4) guarantees that every pickup node is visited exactly once, while (5) ensures that the pickup and delivery nodes of a request are visited by the same agent. The combination of (4) and (5) also warrants unique visits to delivery nodes. The agent flow conservation at pickup and delivery nodes is addressed by (6) and (7), respectively. All illegal travel arcs, such as re-entering initial nodes or exiting final depots, are omitted with (8). Lastly, the combination of (8)–(9) guarantees that an active agent can only leave its starting position to directly visit a pickup node, whereas (10) ensures that an inactive agent, i.e., with no assigned requests, does not traverse the network.

$$t_{p^r} \geq \sum_{k \in K} (\delta^k + c_{v^k p^r}) x_{v^k p^r}^k, \quad \forall r \in R \quad (11)$$

$$t_j \geq t_i + s_i + c_{ij} - M \left(1 - \sum_{k \in K} x_{ij}^k \right), \quad \forall i, j \in S', i \neq j \quad (12)$$

$$T \geq t_{d^r} + s_{d^r} + \sum_{j \in H_f} \sum_{k \in K} c_{d^r j} x_{d^r j}^k, \quad \forall r \in R \quad (13)$$

$$t_{d^r} - t_{p^r} - s_{p^r} \geq c_{p^r d^r}, \quad \forall r \in R \quad (14)$$

$$t_{d^r} - t_{p^r} - s_{p^r} \leq \rho c_{p^r d^r}, \quad \forall r \in R \quad (15)$$

$$e_{p^r} \leq t_{p^r} \leq l_{p^r} + \tau_{p^r}, \quad \forall r \in R_s \quad (16)$$

$$e_{p^r} - \tau_{p^r} \leq t_{p^r} \leq l_{p^r} + \tau_{p^r}, \quad \forall r \in R_n \quad (17)$$

$$e_{p^r} - \omega \leq t_{p^r}, \quad \forall r \in R_n \quad (18)$$

$$\sum_{i \in V'} \sum_{j \in V'} c_{ij} x_{ij}^k \leq D^k, \quad \forall k \in K \quad (19)$$

For temporal constraints, (11) propagates the time of earliest availability from agent starting positions to the first visited pickup nodes, while (12) ensures continuity between consecutively visited stations. By enforcing only lower bounds on the timing propagation constraints, we implicitly permit agents to wait before starting service, where waiting times between consecutive nodes i, j can be inferred with $t_j - (t_i + s_i + c_{ij})$. This

approach is efficient as it reduces the number of constraints and variables, albeit with a slight decrease in readability. Equation (13) determines the mission duration, wherein reducing it to $T \geq t_{dr} + s_{dr}, \forall r \in R$ accommodates the OVRP objective by excluding travel times to final depots from the objective calculation. Constraint (14) guarantees pickup before delivery, while (15) restricts maximum ride times using the configurable user-satisfaction parameter ρ . For TW violations, we distinguish scheduled and new requests with (16) and (17)–(18), respectively. As previously explained, MILP1 imitates energy limitations using travel time, such that (19) restricts the driving duration of each agent based on its worst-case battery discharge. For OVRPs, we emphasize that route segments to H_f remain included in (19), despite their exclusion from the objective calculation in (13). This design choice promotes operational robustness by guaranteeing that agents have sufficient energy to reach a depot after completing their final delivery.

$$z_j \geq z_i + q_j - Q \left(1 - \sum_{k \in K} x_{ij}^k \right), \quad \forall i, j \in S', i \neq j \quad (20)$$

$$q_i \leq z_i \leq \sum_{k \in K} Q^k \sum_{j \in S' \setminus \{i\}} x_{ij}^k, \quad \forall i \in S^p \quad (21)$$

$$z_i \geq 0, \quad \forall i \in S^d \quad (22)$$

For the capacity constraints, we accelerate convergence by utilizing the sufficiently large constant $Q = 2 \max_{k \in K} Q^k$ for constraint deactivation. Herein, (20) ensures load continuity between consecutive service stations, excluding upper bounds as solutions with lower loads dominate higher loads. Moreover, we enforce agent capacity limits at pickup and delivery with (21) and (22), respectively. The efficient formulation in (21) selects the capacity of the currently visiting agent as the active limit, despite having heterogeneous agents. Since passengers are restricted to entry at pickup stations, upper bounds are not necessary for delivery stations in (22).

To summarize, MILP1 minimizes the objective (3) and adheres to the constraints (4)–(22); it accommodates closed and open objectives by modifying (13). The model is advantageous due to its low complexity, but it significantly restricts the maximum driving duration. Moreover, the model necessitates fulfilling all requests, which may be infeasible depending on the agent availability and the number and demands of requests.

4.2 MILP2: Hard Time Windows with Extended SoC

In the second model, we enforce hard TWs, which may render the problem infeasible; This is mitigated by adopting a selective variant capable of rejecting user requests [14]. Furthermore, we employ the extended tuple $(v^k, \delta^k, Q^k, \underline{\sigma}^k, \sigma^k, \alpha^k, \beta^k)$ to permit longer travel routes. In addition to the variables x_{ij}^k, t_i, T, z_i defined in MILP1, we introduce:

- $y_r = \begin{cases} 1, & \text{if request } r \in R \text{ is accepted (assigned to an agent),} \\ 0, & \text{otherwise.} \end{cases}$
- $\phi_i \in [0, 1]$: auxiliary SoC value upon arrival at service node $i \in S'$

where $t_i = 0$ if node $i \in S'$ is not visited, i.e., belongs to a rejected request. We propose an objective function that penalizes the rejection of requests:

$$\min T + \sum_{r \in R} (\epsilon (t_{p^r} + t_{d^r}) + \eta (1 - y_r)) \quad (23)$$

where $\eta \gg \epsilon > 0$ strongly penalizes request rejection without imposing a lexicographic objective. We highlight that an accepted request satisfies $y_r = 1$ and has the arrival times t_{p^r}, t_{d^r} ; otherwise, these values shall be equal to zero. Since these times are guaranteed to be non-negative with $t_i \geq 0, i \in S'$, it is sufficient to add the constraint:

$$t_{p^r} \leq M y_r, \quad t_{d^r} \leq M y_r, \quad \forall r \in R \quad (24)$$

For request assignment and flow conservation, we adopt the constraints (5)–(10) from MILP1, in addition to:

$$y_r = 1, \quad \forall r \in R_s \quad (25)$$

$$\sum_{k \in K} \sum_{i \in (H_0 \cup S') \setminus \{p^r\}} x_{ip^r}^k = y_r, \quad \forall r \in R \quad (26)$$

maintaining prior acceptance commitments for scheduled requests and ensuring that exactly one agent enters the pickup node of an accepted request. When a request is rejected $y_r = 0$, its corresponding nodes are not visited. We employ the timing constraints (11) and (12) from MILP1, and introduce the modified constraints:

$$T \geq t_{d^r} + s_{d^r} + \sum_{j \in H_f} \sum_{k \in K} c_{d^r j} x_{d^r j}^k - M(1 - y_r), \quad \forall r \in R \quad (27)$$

$$t_{d^r} - t_{p^r} - s_{p^r} \geq c_{p^r d^r} - M(1 - y_r), \quad \forall r \in R \quad (28)$$

$$t_{d^r} - t_{p^r} - s_{p^r} \leq \rho c_{p^r d^r} + M(1 - y_r), \quad \forall r \in R \quad (29)$$

$$e_{p^r} - M(1 - y_r) \leq t_{p^r} \leq l_{p^r} + M(1 - y_r), \quad \forall r \in R \quad (30)$$

which are deactivated for rejected requests. Similar to MILP1, (27) can be modified to support the OVRP objective. Note that hard TWs are indiscriminately enforced for new and scheduled requests in (30). For capacity constraints, we similarly adopt (20) and (22) from MILP1, and modify the pickup load limits to be:

$$q_{p^r} y_r \leq z_{p^r} \leq \sum_{k \in K} Q^k \sum_{j \in S' \setminus \{p^r\}} x_{p^r j}^k + Q(1 - y_r), \quad \forall r \in R \quad (31)$$

so that they are deactivated for rejected requests. Finally, we have the SoC constraints:

$$\phi_{p^r} \leq \sum_{k \in K} (\sigma^k - \alpha^k c_{v^k p^r}) x_{v^k p^r}^k + 2 \left(1 - \sum_{k \in K} x_{v^k p^r}^k \right), \quad \forall r \in R \quad (32)$$

$$\phi_j \leq \phi_i - \alpha^k c_{ij} - \beta^k c_{ij} z_i + 2 (1 - x_{ij}^k), \quad \forall i, j \in S', i \neq j, k \in K \quad (33)$$

$$\sum_{k \in K} \underline{\sigma}^k \sum_{i \in (H_0 \cup S') \setminus \{j\}} x_{ij}^k \leq \phi_j \leq 1, \quad \forall j \in S' \quad (34)$$

$$\phi_i - \sum_{k \in K} \sum_{j \in H_f} \alpha^k c_{ij} x_{ij}^k \geq \sum_{k \in K} \underline{\sigma}^k \sum_{j \in H_f} x_{ij}^k, \quad \forall i \in S^d \quad (35)$$

Owing to energy parameterization, the constant value 2 suffices to deactivate the SoC propagation constraints. Constraint (32) models the SoC change upon direct travel from H_0 to S^p , while (33) represents the load-dependent SoC drop between consecutive stations. Constraints (34)–(35) establish the operational limits and prevent mid-route battery depletion, with an efficient formulation for activating the adequate SoC limit. We emphasize that including route segments to H_f in (35) for OVRPs ensures that agents have sufficient energy to return to a depot after route completion. Also, we highlight that the auxiliary SoC variables ϕ_i may attain lower values than expected, as (32)–(33) only impose upper-bound propagation. Nevertheless, these values preserve route feasibility and do not impact the objective, whereas including the complementary lower bounds increases the problem size; they are therefore omitted.

Compared to MILP1, this model permits the generation of longer travel routes, albeit at the cost of increased complexity and potentially slower response times. In addition, its selectivity enables the rejection of new requests in case of insufficient resources; provided that scheduled requests and updated agent states preserve feasibility of the remaining plan from the previous planning epoch, MILP2 is guaranteed to find a feasible solution, regardless of the nature of new requests.

4.3 MILP3: Soft Time Windows with Extended SoC

We combine the features of both approaches to obtain a more comprehensive and realistic model, with request selectivity, soft TWs, and improved SoC representation; an overview of MILP3 is presented in Table 2. Herein, we employ the objective function:

$$\min \quad T + \sum_{r \in R} (\epsilon (t_{p^r} + t_{d^r}) + \zeta \tau_{p^r} + \eta (1 - y_r)) \quad (36)$$

where $\eta \gg \zeta > \epsilon > 0$. Similar to t_i , we fix the pickup slack times of rejected requests to zero with:

$$\tau_{p^r} \leq M y_r, \quad \forall r \in R \quad (37)$$

which, in turn, mandates adapting the TW restrictions as follows:

$$e_{p^r} - M(1 - y_r) \leq t_{p^r} \leq l_{p^r} + \tau_{p^r} + M(1 - y_r), \quad \forall r \in R_s \quad (38)$$

$$e_{p^r} - \tau_{p^r} - M(1 - y_r) \leq t_{p^r} \leq l_{p^r} + \tau_{p^r} + M(1 - y_r), \quad \forall r \in R_n \quad (39)$$

$$e_{p^r} - \omega - M(1 - y_r) \leq t_{p^r}, \quad \forall r \in R_n \quad (40)$$

Combined with (25), the model can be converted into a non-selective variant with:

$$y_r = 1, \quad \forall r \in R_n \quad (41)$$

To summarize, MILP3 has the objective (36) and is subject to the constraints:

- **Objective calculation:** (24) and (37).

- **Request assignment and flow conservation:** (5)–(10), (25), and (26). If desired, incorporating (41) forces the acceptance of all requests.
- **Temporal integrity:** (11)–(12), (27)–(29), and (38)–(40).
- **Passenger load:** (20), (22), and (31).
- **Energy consumption and minimum SoC:** (32)–(35).

Table 2 Sets, parameters, and variables used in MILP3

Symbol	Description
<i>Model sets</i>	
K	heterogeneous fleet agents
R_n	newly received passenger requests that can be freely accepted or rejected
R_s	scheduled requests whose prior acceptance commitments must be preserved
R	complete set of requests $R = R_n \cup R_s$ currently considered in the routing problem
H_0	virtual initial nodes, agent starting positions $H_0 = \{v^k : k \in K\}$
S^p	virtual, request-specific pickup nodes $S^p = \{p^r : r \in R\}$
S^d	virtual, request-specific delivery nodes $S^d = \{d^r : r \in R\}$
S'	combined set of virtual pickup and delivery nodes $S' = S^p \cup S^d$
H_f	virtual final depots, copies of physical hubs
V'	complete set of virtual network nodes $V' = H_0 \cup S' \cup H_f$
E'	complete set of virtual network arcs $E' = V' \times V'$
<i>Agent parameters</i>	
$v^k \in H_0$	virtual starting node of agent $k \in K$ at its earliest availability
$\delta^k \in \mathbb{R}_0^+$	initial delay, time of earliest availability at v^k
$\sigma^k \in [\sigma^k, 1]$	initial SoC at v^k after δ^k has elapsed
$Q^k \in \mathbb{Z}^+$	agent capacity, maximum passenger load of agent $k \in K$
$\underline{\sigma}^k \in (0, 1]$	minimum operational SoC to avoid mid-course battery depletion
$\alpha^k \in \mathbb{R}^+$	load-free, normalized battery discharge rate per minute of driving
$\beta^k \in \mathbb{R}^+$	load-dependent, normalized battery discharge rate per minute of driving
<i>Network, model, and cost parameters</i>	
$c_{ij} \in \mathbb{R}_0^+$	arc travel time between $i, j \in V'$, with $c_{ii} = 0$
$s_i \in \mathbb{R}_0^+$	required service time at node $i \in S'$
$e_i, l_i \in \mathbb{R}_0^+$	bounds of the desired TW at pickup node $i \in S^p$, with $l_i > e_i$
$\omega \in \mathbb{R}^+$	fixed TW length, with $l_i = e_i + \omega$ for $i \in S^p$
$\rho \in [1, \infty)$	configurable user-satisfaction parameter for the maximum ride time
q_i	number of passengers at $i \in S'$, with $q_i \in \mathbb{Z}^+$, $\forall i \in S^p$, and $q_i \in \mathbb{Z}^-$, $\forall i \in S^d$, and $q_{p^r} = -q_{d^r}$, $\forall r \in R$
$Q \in \mathbb{Z}^+$	sufficiently large constant to deactivate load constraints, $Q = 2 \max_{k \in K} Q^k$
$M \in \mathbb{R}^+$	sufficiently large constant to deactivate time constraints, derived from the rolling-horizon planning schedule
$\epsilon \in \mathbb{R}^+$	small penalty parameter to minimize request arrival times
$\zeta \in \mathbb{R}^+$	relatively large penalty parameter to minimize TW violations, with $\zeta > \epsilon$
$\eta \in \mathbb{R}^+$	considerably large penalty parameter to minimize request rejection, with $\eta \gg \zeta > \epsilon$
<i>Model variables</i>	
$x_{ij}^k \in \{0, 1\}$	1 if agent $k \in K$ travels from $i \in V'$ to $j \in V'$; 0 otherwise
$y_r \in \{0, 1\}$	1 if request $r \in R$ is accepted (assigned to an agent); 0 otherwise
$t_i \in \mathbb{R}_0^+$	start of service, arrival time at node $i \in S'$
$\tau_i \in \mathbb{R}_0^+$	slack variable for penalizing pickup TW violations at $i \in S^p$; 0 if TW is not violated
$T \in \mathbb{R}_0^+$	mission duration, maximum route-completion time across all agents
$z_i \in \mathbb{Z}_0^+$	passenger load upon departure from $i \in S'$
$\phi_i \in [0, 1]$	auxiliary SoC value upon arrival at service node $i \in S'$

4.4 Constructive Heuristics for Feasible Initialization

Constructive and greedy methods are often utilized to rapidly identify feasible solutions, which act as incumbents for warm starting exact methods or as initial solutions for iterative heuristic and metaheuristic approaches [17, 23, 32]. In the sequel, we introduce a common solution representation and preprocessing procedure, which we later use for our MILPs and ALNS.

A solution Ω defines one journey for each agent $k \in K$, comprising:

$$\Omega := \{(\mathcal{V}^k, \mathcal{T}^k, \mathcal{Z}^k, \Phi^k) : k \in K\}, \quad (42)$$

where $\mathcal{V}^k$ is the route, i.e., ordered sequence of visited nodes, $\mathcal{T}^k$ the associated timing schedule, $\mathcal{Z}^k$ the propagated passenger loads, and Φ^k the corresponding SoC values. An agent route $\mathcal{V}^k = (n_0^k, \dots, n_{m_k}^k)$ satisfies $n_0^k = v^k$, $n_i^k \in S'$ for $0 < i < m_k$, and $n_{m_k}^k = h_f^k \in H_f$, i.e., every route starts at an initial depot and terminates at a final depot. Active agents are denoted by $K_a := \{k \in K : \mathcal{V}^k \cap S' \neq \emptyset\}$, while inactive agents have dummy routes $\mathcal{V}^k = (v^k, h_f^k)$, $k \notin K_a$. As previously mentioned, we exclude the cost of final arcs from the objective calculation of OVRPs, while retaining their energy feasibility constraints. Furthermore, two disjoint request sets accompany a complete solution, namely the accepted requests $R_a \subseteq R$ and rejected requests $R_r \subseteq R$, with $R_a \cap R_r = \emptyset$ and $R_a \cup R_r = R$. The pickup and delivery nodes of every accepted request appear exactly once in the solution, within the same agent route, and satisfy precedence conditions; the nodes of every rejected request $r \in R_r$ do not appear in any agent route.

With a slight abuse of notation, we introduce the arrival state $(t_{arr,i}^k, z_{arr,i}^k, \phi_{arr,i}^k)$ and departure state $(t_{dep,i}^k, z_{dep,i}^k, \phi_{dep,i}^k)$ at every visited node for exact value propagation. Arrival states are undefined at v^k , and departure states are similarly undefined at h_f^k . Moreover, we explicitly define waiting variables $w_i^k \geq 0$ at every initial or service node $i \in (H_0 \cup S') \cap \mathcal{V}^k$. Dynamics propagation starts at an initial node with:

$$t_{dep,v^k}^k = \delta^k + w_{v^k}^k, \quad z_{dep,v^k}^k = 0, \quad \phi_{dep,v^k}^k = \sigma^k, \quad (43)$$

then, every pair of consecutive nodes (i, j) in $\mathcal{V}^k$ yields:

$$t_{arr,j}^k = t_{dep,i}^k + c_{ij}, \quad (44)$$

$$z_{arr,j}^k = z_{dep,i}^k, \quad (45)$$

$$\phi_{arr,j}^k = \phi_{dep,i}^k - c_{ij} (\alpha^k + \beta^k z_{dep,i}^k), \quad (46)$$

until reaching h_f^k . At a visited service node $i \in S' \cap \mathcal{V}^k$, service starts at $t_{arr,i}^k + w_i^k$, and the node departure state is computed with:

$$t_{dep,i}^k = t_{arr,i}^k + w_i^k + s_i, \quad z_{dep,i}^k = z_{arr,i}^k + q_i, \quad \phi_{dep,i}^k = \phi_{arr,i}^k. \quad (47)$$

Accordingly, our explicit solution representation can be directly mapped to the MILP formulations, with $t_i = t_{arr,i}^k + w_i^k$, $z_i = z_{dep,i}^k$, and $\phi_i = \phi_{arr,i}^k$ for agent k

visiting node i . As discussed, (47) assumes that waiting and service operations do not contribute to energy consumption.

In the preprocessing stage, we validate basic problem feasibility and compile a base plan, where the active strategy index $\kappa \in \{\text{MILP1}, \text{MILP2}, \text{MILP3}, \text{ALNS}\}$ dictates the desired feasibility conditions. If all requests are new, we ensure that all agents possess sufficient energy to reach a depot, establishing a base plan of dummy routes and leaving all requests unassigned. Alternatively, if a partial plan is inherited from a previous planning cycle, we update it using the current agent states, propagating all journeys and asserting that all scheduled requests can still be satisfied. In case a partial plan is accompanied by a mixture of scheduled and new requests, we (i) validate the previous plan and (ii) compile a feasible base plan, where all scheduled requests are met and all new requests remain unassigned. Herein, (43)–(47) are utilized for dynamic state propagation, and an infeasible journey terminates the preprocessing stage with a failure flag, indicating an unsolvable problem.

Algorithm 1 Preprocessing and base plan generation

Require: Strategy κ ; updated fleet K ; request sets R_s, R_n ; optional partial plan Ω_p
Ensure: Base plan Ω_b ; accepted requests R_a ; unassigned requests R_u ; success flag f

```

1: Initialize  $\Omega_b \leftarrow \emptyset$ ,  $R_a \leftarrow \emptyset$ , and  $R_u \leftarrow R_n$ 
2: if  $\Omega_p$  is available then
3:   Update  $\Omega_p$  using current agent states and propagate all journeys
4:    $\Omega_b \leftarrow \Omega_p$  and  $R_a \leftarrow$  requests assigned in  $\Omega_b$ 
5:   if  $\Omega_b$  is infeasible for  $\kappa$  or  $R_s \not\subseteq R_a$  then
6:     return  $(\Omega_b, R_a, R_u, 0)$ 
7:   end if
8: else
9:   if  $R_s \neq \emptyset$  then
10:    return  $(\emptyset, \emptyset, R_u, 0)$ 
11:   end if
12:   for  $k \in K$  do
13:      $w_{v^k}^k \leftarrow 0$ ,  $h_f^k \leftarrow \operatorname{argmin}_{h \in H_f} c_{v^k h}$ , and  $\mathcal{V}^k \leftarrow (v^k, h_f^k)$ ; propagate journey states
14:     if the journey is infeasible for  $\kappa$  then
15:       return  $(\Omega_b, \emptyset, R_u, 0)$ 
16:     end if
17:     Add the journey of agent  $k$  to  $\Omega_b$ 
18:   end for
19: end if
20: return  $(\Omega_b, R_a, R_u, 1)$ 

```

In the second stage, all constructive methods follow the greedy completion procedure in Algorithm 2. New requests are sorted by latest pickup time l_{pr} in increasing order, i.e., prioritizing the earliest deadline, with the higher demand q^r as a tie-breaker. To accelerate the processing of each request, the strategy-specific function $\mathcal{C}_\kappa(r, \Omega)$ identifies feasible request insertions and generates candidate solutions with their corresponding objective values. Here, the candidate with the smallest value is selected; if no feasible candidate exists, MILP1 terminates with a failure flag, whereas the selective MILP2, MILP3, and ALNS strategies reject the request and proceed normally.

Algorithm 2 Greedy completion of a feasible base plan

Require: Strategy κ ; base plan Ω_b ; sets R_a, R_u ; objective function $\mathcal{F}_\kappa(\cdot)$

Ensure: Completed solution Ω ; rejected requests R_r ; success flag f

```
1:  $\Omega \leftarrow \Omega_b, R_r \leftarrow \emptyset$ , and sort  $\hat{R}_u \leftarrow R_u$  by smallest  $l_{p^r}$  with largest  $q^r$  as tie-breaker
2: for  $u \in \hat{R}_u$  do
3:    $\mathcal{C}_u \leftarrow \mathcal{C}_\kappa(u, \Omega)$ 
4:   if  $\mathcal{C}_u \neq \emptyset$  then
5:     Select  $(\hat{\Omega}^*, \hat{\mathcal{F}}_\kappa^*) \in \mathcal{C}_r$  with minimum  $\hat{\mathcal{F}}_\kappa^*$ 
6:      $\Omega \leftarrow \hat{\Omega}^*, R_a \leftarrow R_a \cup \{r\}$ , and  $R_u \leftarrow R_u \setminus \{r\}$ 
7:   else if  $\kappa = \text{MILP1}$  then
8:     return  $(\Omega, R_r, 0)$ 
9:   else
10:     $R_r \leftarrow R_r \cup \{u\}$  and  $R_u \leftarrow R_u \setminus \{u\}$ 
11:   end if
12: end for
13: return  $(\Omega, R_r, 1)$ 
```

For MILPs, $\mathcal{C}_\kappa$ considers one direct insertion per capacity-compatible agent, where we extend the current journey by replacing its final depot h_f^k with $(p^r, d^r, \hat{h}_f^k)$, such that $\hat{h}_f^k = \arg\min_{h \in H_f} c_{drh}$. ALNS employs a more comprehensive strategy, which is later explained in Section 4.5.1. The modified journey nodes are forward-propagated and validated against the active κ feasibility conditions. Specifically, MILP1 applies the remaining driving duration D^k , while MILP2, MILP3, and ALNS apply the extended SoC conditions. Additionally, MILP2 enforces hard TWs, whereas maximum ride time, precedence, and capacity constraints apply to all formulations. During propagation, we introduce waiting to omit early service starting before e_{p^r} for ALNS, which is a common approach adopted from existing literature and adequately justified in the next section. Provided that a solution Ω is successfully constructed, we can directly utilize it to populate the relevant MILP variables x_{ij}^k , t_i , and z_i , as well as ϕ_i for MILP2 and MILP3. For the selective variants, we additionally set $y_r = 1$ for accepted requests $r \in R_a$ and $y_r = 0$ for rejected requests $r \in R_r$, with zero temporal service-node variables and feasible SoC variables $\phi_i = 1, i \in \{p^r, d^r\}, r \in R_r$. Inactive agents with dummy routes $\mathcal{V}^k = (v^k, h_f^k)$ are safely ignored, setting their corresponding travel arcs $x_{ij}^k = 0, \forall i, j \in V'$ to satisfy (10). Finally, an exact solution can be constructed from a generated MILP solution using forward propagation along the determined route, wherein waiting times can be inferred from the t_i variables.

4.5 Adaptive Large Neighborhood Search (ALNS)

Our developed ALNS adheres to the objective, constraints, and dynamics propagation of MILP3, with the exception that it does not permit early service for new requests; agents arriving at a request pickup consistently wait until its TW opens $t = e_{p^r}$, which may yield suboptimal results. Figure 2 illustrates that waiting at an upstream node propagates tardiness through the remaining nodes, which could have been mitigated by violating the earliest pickup TW e_{p^1} . Nevertheless, this is a common simplification when employing ALNS, cf. [12, 27, 32, 35], since, otherwise, a scheduling optimization problem arises that significantly complicates the implementation.

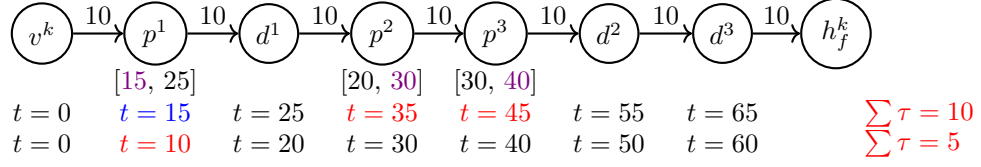


Fig. 2 Illustrative example to emphasize the suboptimality of omitting early service. Constant travel times $c_{ij} = 10$ min and zero service times are assumed. Waiting at p^1 to satisfy $e_{p^1} = 15$ increases tardiness and incurs a higher cumulative penalty downstream.

Waiting and tardiness at the pickup node of an accepted request are computed with:

$$w_{p^r}^k = \max \{0, e_{p^r} - t_{arr,p^r}^k\}, \quad \tau_{p^r} = \max \{0, t_{arr,p^r}^k + w_{p^r}^k - l_{p^r}\}, \quad (48)$$

with $t_{p^r} = t_{arr,p^r}^k + w_{p^r}^k$ for the objective calculation. Unless required by (48), waiting is set to zero at all remaining route nodes. As mentioned, temporal variables of all rejected requests $t_{p^r} = t_{d^r} = \tau_{p^r} = 0, r \in R_r$ are set to zero. Correspondingly, $y_r = 1$ for $r \in R_a$ and $y_r = 0$ for $r \in R_r$. Analogous to (36), we calculate the objective function of a partial or complete solution with:

$$\mathcal{F}(\Omega) := T + \sum_{r \in R} (\epsilon(t_{p^r} + t_{d^r}) + \zeta \tau_{p^r} + \eta(1 - y_r)), \quad (49)$$

and compute the mission duration as $T = \max_{k \in K_a} t_{arr,h_f^k}^k$ for the closed objective and $T = \max_{r \in R_a} \{t_{d^r} + s_{d^r}\}$ for OVRP. Herein, $T = 0$ if $K_a = \emptyset$ or $R_a = \emptyset$, respectively.

In addition to request pairing and precedence constraints, which are enforced by the journey representation and insertion procedures, every ALNS solution satisfies:

$$R_s \subseteq R_a, \quad (50)$$

$$0 \leq z_{dep,i}^k \leq Q^k, \quad \forall i \in S' \cap \mathcal{V}^k, \quad k \in K, \quad (51)$$

$$t_{d^r} - t_{p^r} - s_{p^r} \leq \rho c_{p^r d^r}, \quad \forall r \in R_a, \quad (52)$$

$$\phi_{arr,i}^k \geq \underline{\sigma}^k, \quad \forall i \in (S' \cup H_f) \cap \mathcal{V}^k, \quad k \in K. \quad (53)$$

Note that an ALNS solution is feasible by design; only feasible request insertions are considered and infeasible candidates are discarded before entering the search. If the destroy operation removes scheduled requests that are not successfully restored during the repair operation, the resulting candidate solution is likewise discarded and the current solution remains unchanged.

4.5.1 ALNS Solution Construction

Inspired by [19], we employ a feasibility-driven approach to identify admissible request insertions both during initial solution construction and when repairing a partially destroyed ALNS solution, thereby reducing search runtime. Given a route-feasible, propagated partial solution Ω and an unassigned request $u \in R_u$, this procedure yields

the feasible candidate solutions and their accompanying objective values:

$$\mathcal{C}_{ALNS}(u, \Omega) := \left\{ \left(\widehat{\Omega}_{ab}^{uk}, \widehat{\mathcal{F}}_{ab}^{uk} \right) : k \in K_q, 0 \leq a \leq b < m_k, \widehat{\Omega}_{ab}^{uk} \text{ is feasible} \right\}, \quad (54)$$

where $K_q := \{k \in K : q^u \leq Q^k\}$ represents capacity-compatible agents. The feasible candidate $\widehat{\Omega}_{ab}^{uk}$ denotes the updated solution after the insertion pair (a, b) places request u in the journey of agent k , including all propagated timing, load, and SoC states.

When first invoked, we cache the original solution objective $\mathcal{F}_0 := \mathcal{F}(\Omega)$ and mission time T_0 . Then, we cache agent-specific values when iterating over $k \in K_q$; for each k , we record its exact states, waiting times, request positions, arc-energy and node-objective contributions. Moreover, we identify its terminal SoC slack ξ^k and the ride-time slack θ^r for each of its assigned requests with:

$$\xi^k := \phi_{arr, h_f^k}^k - \underline{\sigma}^k, \quad (55)$$

$$\theta^r := \rho c_{p^r d^r} - (t_{d^r} - t_{p^r} - s_{p^r}) \geq 0, \quad r \in R_a, \{p^r, d^r\} \in S' \cap \mathcal{V}^k, \quad (56)$$

respectively. For the agent route $\mathcal{V}^k = (n_0^k, \dots, n_{m_k}^k)$, an insertion pair (a, b) places p^u after n_a^k and d^u after n_b^k , with $0 \leq a \leq b < m_k$. Herein, we distinguish the unchanged prefix, modified segment, and stale suffix:

$$\mathcal{V}_{0:a}^k := (n_0^k, \dots, n_a^k), \quad \widehat{\mathcal{S}}_{ab}^{uk} := (p^u, n_{a+1}^k, \dots, n_b^k, d^u), \quad \mathcal{V}_{b+1:m_k}^k := (n_{b+1}^k, \dots, n_{m_k}^k), \quad (57)$$

respectively, such that the sequence $(n_{a+1}^k, \dots, n_b^k)$ is empty for $a = b$. Accordingly, the candidate route is:

$$\widehat{\mathcal{V}}_{ab}^{uk} := \begin{cases} \left(\mathcal{V}_{0:a}^k, \widehat{\mathcal{S}}_{ab}^{uk}, \mathcal{V}_{b+1:m_k}^k \right), & b < m_k - 1, \\ \left(\mathcal{V}_{0:a}^k, \widehat{\mathcal{S}}_{a, m_k-1}^{uk}, \widehat{h}_f^k \right), & b = m_k - 1, \end{cases} \quad (58)$$

where $b = m_k - 1$ triggers the final depot re-optimization from the newly inserted delivery d^u with $\widehat{h}_f^k := \operatorname{argmin}_{h \in H_f} c_{d^u h}$. Since we place both nodes in the same route and enforce $a \leq b$, we inherently satisfy request pairing, same-agent assignment, and precedence. For load and SoC constraints, we denote the energy consumed on arc (i, j) when departing with load z by:

$$\mathcal{E}_{ij}^k(z) := c_{ij} (\alpha^k + \beta^k z), \quad (59)$$

and only consider a pickup position a if agent k can carry u and reach its pickup:

$$z_{dep, n_a^k}^k + q^u \leq Q^k, \quad \phi_{dep, n_a^k}^k - \mathcal{E}_{n_a^k p^u}^k(z_{dep, n_a^k}^k) \geq \underline{\sigma}^k, \quad (60)$$

where satisfying these conditions appends p^u to the prefix, propagating states with (44)–(47). Starting with $b = a$, we incrementally extend the modified segment $\widehat{\mathcal{S}}_{ab}^{uk}$ by

original nodes, wherein a capacity, local SoC, or request ride-time violation in this segment terminates the delivery scan because every subsequent position retains the infeasible segment. For ride-time determination, we denote the set of on-board requests when departing from position i by:

$$\mathcal{B}_i^k := \{r \in R_a : \text{pos}^k(p^r) \leq i < \text{pos}^k(d^r), \{p^r, d^r\} \in S' \cap \mathcal{V}^k\}, \quad (61)$$

and initialize $\widehat{\mathcal{B}} \leftarrow \mathcal{B}_a^k \cup \{u\}$ after pickup insertion. The set is updated at every visited service node j belonging to request $r \in R_a$ according to:

$$\widehat{\mathcal{B}} \leftarrow \begin{cases} \widehat{\mathcal{B}} \cup \{r\}, & j = p^r \wedge a < \text{pos}^k(p^r) \leq b, \\ \widehat{\mathcal{B}} \setminus \{r\}, & j = d^r \wedge r \in \widehat{\mathcal{B}}, \text{ after ensuring } \widehat{\theta}^r \geq 0, \\ \widehat{\mathcal{B}}, & \text{otherwise,} \end{cases} \quad (62)$$

where visiting a pickup in the changed segment affixes its request, while a delivery first validates the updated request ride-time before removing it. The investigated request u is separately handled; its ride time is evaluated directly at d^u , then removed from $\widehat{\mathcal{B}}$.

The extra distance traveled while carrying u provides a monotone lower bound on the additional energy consumption. Herein, we define the on-board distance:

$$\lambda_{ab}^{uk} := \begin{cases} c_{p^u d^u}, & b = a, \\ c_{p^u n_{a+1}^k} + \sum_{\ell=a+1}^{b-1} c_{n_\ell^k n_{\ell+1}^k} + c_{n_b^k d^u}, & b > a, \end{cases} \quad \underline{\Delta \mathcal{E}}_{ab}^{uk} := \beta^k q^u \lambda_{ab}^{uk}, \quad (63)$$

where triangle inequality implies $\lambda_{a,b+1}^{uk} \geq \lambda_{ab}^{uk}$. Hence, $\underline{\Delta \mathcal{E}}_{ab}^{uk} > \xi^k$ safely terminates the delivery scan, since the additional load alone exhausts the terminal SoC slack.

For a complete candidate, let $\mathcal{A}_{ab}^k$ contain the replaced original arcs and $\widehat{\mathcal{A}}_{ab}^{uk}$ their candidate counterparts, including the final-depot arcs when $b = m_k - 1$. The exact additional energy consumption and remaining terminal SoC slack are given by:

$$\Delta \mathcal{E}_{ab}^{uk} := \sum_{(i,j) \in \widehat{\mathcal{A}}_{ab}^{uk}} \mathcal{E}_{ij}^k(z_{dep,i}^k) - \sum_{(i,j) \in \mathcal{A}_{ab}^k} \mathcal{E}_{ij}^k(z_{dep,i}^k), \quad (64)$$

$$\widehat{\xi}_{ab}^{uk} := \xi^k - \Delta \mathcal{E}_{ab}^{uk}, \quad (65)$$

respectively. After departing from d^u , the load profile and arc-energy consumption from the original nodes apply, with shifting the SoC of every suffix node by $-\Delta \mathcal{E}_{ab}^{uk}$ as no charging occurs. We emphasize that local feasibility within the changed segment together with $\widehat{\xi}_{ab}^{uk} \geq 0$ guarantee SoC feasibility throughout the suffix without propagating its SoC states. However, arrival times can be stale, which we now investigate.

After connecting the changed segment to the suffix, we obtain the exact arrival time at $b+1$ from d^u through propagation. Consequently, the insertion-induced, service-start temporal shift satisfies:

$$\Delta t_j^k := \widehat{t}_j - t_j = \max\{0, \Delta t_i^k - w_j^k\}, \quad j \in S', \quad (66)$$

where the nonnegative shift follows from triangle inequality. For every impacted request r , its propagated ride-time slack is:

$$\hat{\theta}^r = \theta^r - (\Delta t_{dr}^k - \Delta t_{pr}^k), \quad (67)$$

where $\Delta t_{pr}^k = 0$ if its pickup precedes the changed segment, while the ride time of u is evaluated directly. Temporal propagation concludes when waiting absorbs the shift, i.e., $\Delta t_j^k = 0$. Thus, requests occurring completely after j or, naturally, within the unchanged prefix remain unaffected. A ride-time violation in the prefix or for u itself terminates the delivery scan, whereas a candidate-specific suffix violation discards only the current candidate. Likewise, violating the monotone lower bound (63) terminates the scan, while an exact terminal SoC violation only discards the current candidate.

For compactness, we denote the objective contribution of a visited request node by:

$$g_i(t) := \begin{cases} \epsilon t + \zeta \max\{0, t - l_i\}, & i \in S^p, \\ \epsilon t, & i \in S^d, \end{cases} \quad (68)$$

since early service is omitted. Let $\mathcal{I}_{ab}^{uk}$ contain the original request nodes, whose service-start times changed due to the request insertion. Since u is unassigned in Ω , $\mathcal{F}_0$ includes its rejection penalty. The objective adjustment and absolute candidate objective are:

$$\Delta \mathcal{F}_{ab}^{uk} := -\eta + g_{p^u}(\hat{t}_{p^u}) + g_{d^u}(\hat{t}_{d^u}) + \sum_{i \in \mathcal{I}_{ab}^{uk}} (g_i(\hat{t}_i) - g_i(t_i)) + \Delta T_{ab}^{uk}, \quad (69)$$

$$\hat{\mathcal{F}}_{ab}^{uk} := \mathcal{F}_0 + \Delta \mathcal{F}_{ab}^{uk}, \quad (70)$$

respectively, where ΔT_{ab}^{uk} denotes the potential increase in mission duration. We emphasize that under triangle inequality and nearest-depot selection, an insertion cannot reduce the completion value of an impacted journey. If waiting absorbs the temporal shift before the final depot (for closed routing) or before the final service node (for open routing), then $\Delta T_{ab}^{uk} = 0$. Otherwise, the propagated final-hub arrival or final-delivery completion is compared with the original duration T_0 , respectively. Hence, the candidate objective is updated only through the inserted and temporally shifted nodes.

In terms of performance, our caching requires $O(|R| + |K| + \sum_{k \in K_q} m_k)$ operations for a journey with m_k insertion arcs, after which a candidate propagates $L_{ab}^{uk} \leq m_k$ suffix nodes before its temporal shift is absorbed. The complete enumeration requires:

$$O\left(|R| + |K| + \sum_{k \in K_q} \left(m_k + \sum_{0 \leq a < m_k} (m_k - a) + \sum_{0 \leq a \leq b < m_k} L_{ab}^{uk}\right)\right), \quad (71)$$

with a worst-case runtime of $O\left(\sum_{k \in K_q} m_k^3\right)$ that approaches $O\left(\sum_{k \in K_q} m_k^2\right)$ when the temporal shift is absorbed after a bounded number of nodes. Every retained candidate already contains the propagated solution and its updated objective value; it can be adopted directly in constant time. These insights closely follow the work of [19].

Algorithm 3 Feasible ALNS request insertions

Require: Partial solution Ω ; unassigned request u

Ensure: $\mathcal{C}_{\text{ALNS}}(u, \Omega)$ containing $(\hat{\Omega}, \hat{\mathcal{F}})$

```

1: Initialize  $\mathcal{C}_{\text{ALNS}}(u, \Omega) \leftarrow \emptyset$ ;  $K_q \leftarrow \{k \in K : q^u \leq Q^k\}$ 
2: if  $K_q = \emptyset$  then return  $\mathcal{C}_{\text{ALNS}}(u, \Omega)$ 
3: Cache  $\mathcal{F}_0 \leftarrow \mathcal{F}(\Omega)$  and  $T_0 \leftarrow T$ 
4: for  $k \in K_q$  do
5:   Cache the agent-specific states, contributions, and slacks
6:   for  $a = 0$  to  $m_k - 1$  do
7:     if (60) is violated then continue
8:     Append and propagate  $p^u$  after  $\mathcal{V}_{0:a}^k$ ;  $\hat{\mathcal{B}} \leftarrow \mathcal{B}_a^k \cup \{u\}$ 
9:     for  $b = a$  to  $m_k - 1$  do
10:      if  $b > a$  then
11:        Append  $n_b^k$  to propagated insertion prefix; propagate its states; update  $\hat{\mathcal{B}}$ 
12:        if the modified segment is infeasible then break
13:      end if
14:      if  $\Delta \mathcal{E}_{ab}^{uk} > \xi^k$  then break
15:      Construct  $\hat{\mathcal{V}}_{ab}^{uk}$  according to (58)
16:      Propagate  $d^u$  and the affected suffix; form  $\hat{\Omega}_{ab}^{uk}$  and update  $\hat{\mathcal{B}}$ ,  $\Delta \mathcal{E}_{ab}^{uk}$ , and  $\Delta \mathcal{F}_{ab}^{uk}$ 
17:      if  $\hat{t}_{d^u} - \hat{t}_{p^u} - s_{p^u} > \rho c_{p^u d^u}$  then break
18:      if  $\hat{\Omega}_{ab}^{uk}$  satisfies (51)–(53) then
19:         $\hat{\mathcal{F}}_{ab}^{uk} \leftarrow \mathcal{F}_0 + \Delta \mathcal{F}_{ab}^{uk}$ 
20:        Add  $(\hat{\Omega}_{ab}^{uk}, \hat{\mathcal{F}}_{ab}^{uk})$  to  $\mathcal{C}_{\text{ALNS}}(u, \Omega)$ 
21:      end if
22:    end for
23:  end for
24: end for
25: return  $\mathcal{C}_{\text{ALNS}}(u, \Omega)$ 

```

Table 3 Fundamental quantities used for ALNS solution construction

Symbol	Description
K_q	capacity-compatible agents for the investigated request
$\hat{\mathcal{V}}_{ab}^{uk}$	candidate route by inserting p^u after n_a^k and d^u after n_b^k in agent route $\mathcal{V}^k$
$\mathcal{B}_i^k, \hat{\mathcal{B}}$	on-board requests when k departs from i ; on-board requests after p^u insertion
$\xi^k, \hat{\xi}_{ab}^{uk}$	original and candidate terminal SoC slack, respectively
$\theta^r, \hat{\theta}^r$	original and candidate ride-time slack of assigned request r , respectively
Δt_i^k	insertion-induced temporal shift at original route node i
$\mathcal{A}_{ab}^k, \hat{\mathcal{A}}_{ab}^{uk}$	original and candidate travel arcs (i, j) in the modified segment $\hat{\mathcal{S}}_{ab}^{uk}$, respectively
$\lambda_{ab}^{uk}, \Delta \mathcal{E}_{ab}^{uk}$	on-board distance of u and monotone additional-energy lower bound
$\mathcal{F}_0, \Delta \mathcal{F}, \hat{\mathcal{F}}$	cached objective, candidate adjustment, and updated candidate objective
L_{ab}^{uk}	propagated suffix length before temporal shift is absorbed
$\mathcal{C}_{\text{ALNS}}(u, \Omega)$	complete feasible insertion solutions and their objective values

To summarize, the initial ALNS solution is obtained from Algorithms 1 and 2 with $\kappa = \text{ALNS}$ and $\mathcal{C}_\kappa = \mathcal{C}_{\text{ALNS}}$. Algorithm 3 follows the notation defined in Table 3 and is utilized by all repair operators, where requests without feasible insertions remain unassigned, i.e., are rejected. After the repair operation concludes, we ensure (50). Other than insertion candidate enumeration and final depot re-optimization, we follow

the conventional ALNS framework with adaptive roulette-wheel operator selection and a simulated-annealing acceptance criterion [12, 18, 27, 31].

4.5.2 Destroy Operators

A destroy operator selects $R_d \subseteq R_a$ of size $|R_d| = \min\{|R_a|, \max\{1, \lfloor \check{\mu}|R_a| \rfloor\}\}$, with a randomly selected destruction rate $\check{\mu} \in [0.2, 0.4]$. It then sequentially removes both nodes $\{p^u, d^u\}$ of every $u \in R_d$, propagating the impacted journey immediately after each request removal; this propagation may expedite the service-start times of surviving requests and therefore violate ride-time feasibility. In particular, a surviving pickup p^r may advance farther than its corresponding delivery d^r , increasing the request ride-time beyond the hard limit $\rho c_{p^r d^r}$. We prevent such violations using artificial waiting at the surviving pickup nodes [19], which we explain below. Delivery nodes and the final depot are propagated as early as possible; whenever a removal changes the last service node, the final depot is re-optimized. The destroy operation concludes with $R_a \leftarrow R_a \setminus R_d$ and $R_u \leftarrow R_r \cup R_d$, where R_r represents rejected requests prior to destruction.

Before removing each request $u \in R_d$, which has pickup p^u placed after n_a^k and delivery d^u after n_b^k in the agent journey, we identify the set of requests $\mathcal{R}_t^{uk}$ whose pickup times may advance due to the removal of u :

$$\mathcal{R}_t^{uk} := \{r \in R_a \setminus \{u\} : p^r \in \mathcal{V}_{a+1:m_k-1}^k\}. \quad (72)$$

For each request $r \in \mathcal{R}_t^{uk}$, we cache its service-start time t_{p^r} , tardiness τ_{p^r} , and ride-time slack θ^r . After removing u , forward propagation determines the admissible pickup-time advance ϑ_{p^r} and earliest possible start of service $\hat{t}_{p^r}$ with:

$$\vartheta_{p^r} = \min\{\theta^r, \max\{0, t_{p^r} - \hat{t}_{p^r}\}\}, \quad \hat{t}_{p^r} = \max\{e_{p^r}, t_{arr, p^r}^k\}, \quad (73)$$

respectively, where $\hat{t}_{p^r}$ is inferred from the newly propagated arrival time t_{arr, p^r}^k . Thus, we obtain the resulting service-start time, waiting time, and tardiness using:

$$\hat{t}_{p^r}^* = t_{p^r} - \vartheta_{p^r}, \quad w_{p^r}^k = \max\{0, \hat{t}_{p^r}^* - t_{arr, p^r}^k\}, \quad \tau_{p^r}^* = \max\{0, \hat{t}_{p^r}^* - l_{p^r}\}, \quad (74)$$

respectively. Equivalently, $\tau_{p^r}^* = \max\{0, \tau_{p^r} - \vartheta_{p^r}\}$, while the artificial waiting beyond the earliest feasible schedule is $\hat{t}_{p^r}^* - \hat{t}_{p^r} \geq 0$. Given an initially feasible journey and with travel times satisfying the triangle inequality, the service-start time of a surviving delivery cannot increase; therefore, the ride-time increase is bounded by the pickup advance $\vartheta_{p^r} \leq \theta^r$, which preserves maximum ride-time feasibility.

Furthermore, request removal retains capacity and energy feasibility; removing u reduces the departure load at every surviving node between its pickup and delivery by:

$$\hat{z}_{dep, n_j^k}^k = z_{dep, n_j^k}^k - q^u, \quad a < j \leq b, \quad (75)$$

while the loads at all other surviving nodes remain unchanged. Hence, capacity feasibility remains intact. Moreover, energy feasibility is guaranteed by the load reduction and

the triangle inequality. For $a < b$, removing p^u replaces the arcs (n_a^k, p^u) and (p^u, n_{a+1}^k) with (n_a^k, n_{a+1}^k) , wherein:

$$\mathcal{E}_{n_a^k n_{a+1}^k}^k(z_{dep, n_a^k}^k) \leq \mathcal{E}_{n_a^k p^u}^k(z_{dep, n_a^k}^k) + \mathcal{E}_{p^u n_{a+1}^k}^k(z_{dep, p^u}^k). \quad (76)$$

Similarly, if d^u is not the last service node, removing it replaces the arcs (n_b^k, d^u) and (d^u, n_{b+1}^k) with (n_b^k, n_{b+1}^k) , such that:

$$\mathcal{E}_{n_b^k n_{b+1}^k}^k(z_{dep, n_b^k}^k) \leq \mathcal{E}_{n_b^k d^u}^k(z_{dep, n_b^k}^k) + \mathcal{E}_{d^u n_{b+1}^k}^k(z_{dep, d^u}^k), \quad (77)$$

while, if d^u is the last service node with $b = m_k - 1$, the final depot is re-optimized; traveling directly from n_b^k to the optimized depot cannot consume more energy than traveling through d^u to the previous final depot. Subsequent to removal, every unchanged arc in $\mathcal{A}_{a+1:b}^k$ is traversed with q^u fewer passengers and therefore consumes $\beta^k q^u c_{n_{j-1}^k n_j^k}$ less energy, $a < j \leq b$. Note that the direct case $a = b$ follows analogously by replacing $(n_a^k, p^u, d^u, n_{a+1}^k)$ with (n_a^k, n_{a+1}^k) . Consequently, SoC at every surviving node satisfies:

$$\hat{\phi}_{arr, n_j^k}^k \geq \phi_{arr, n_j^k}^k, \quad a < j < m_k, \quad (78)$$

where, if the final depot remains unchanged, (78) holds for $j = m_k$ as well; otherwise, selecting the nearest final depot naturally preserves energy feasibility. To conclude, every request removal yields a fully propagated journey that remains feasible with respect to request ride-time, capacity, and SoC.

Our implemented destroy operators comprise:

1. **OD1: Random.** Remove requests uniformly at random.
2. **OD2: Random-Zone.** Remove uniformly a physical station represented by the service nodes in $\bigcup_{k \in K} (S' \cap \mathcal{V}^k)$, then randomly remove requests whose pickup or delivery maps to this station. Repeat the process until at least $|R_d|$ are removed.
3. **OD3: Lowest-Demand.** Remove requests in increasing order of demand q^r , randomly breaking ties. Low-demand requests exert relatively lower influence on capacity limitations and load-dependent discharge, thereby facilitating their later reinsertion.
4. **OD4: Worst-Cost.** Remove requests in decreasing order of the objective saving:

$$\Delta \mathcal{F}_r^- := \mathcal{F}_{\text{op}}(\Omega) - \mathcal{F}_{\text{op}}(\Omega \setminus r), \quad (79)$$

where $\mathcal{F}_{\text{op}}$ represents the objective function (49) excluding the rejection penalty and $\Omega \setminus r$ contains the propagated journey after removing p^r and d^r .

5. **OD5: Worst-Waiting.** Remove requests in decreasing order of their pickup waiting-time $\mathcal{W}_r := w_{p^r}^{k(r)}$, randomly breaking ties, where r is assigned to agent $k(r)$.
6. **OD6: Worst-Energy.** Remove requests in decreasing order of their marginal energy consumption:

$$\Delta \mathcal{E}_r^- := \mathcal{E}(\Omega) - \mathcal{E}(\Omega \setminus r), \quad (80)$$

where $\mathcal{E}(\Omega)$ denotes the total energy consumption of all active journeys and $\Omega \setminus r$ is propagated after removing request r .

4.5.3 Repair Operators

A repair operator repeatedly inserts requests from R_u by utilizing the feasible candidate set $\mathcal{C}_{\text{ALNS}}(r, \Omega)$. Scheduled requests in $R_s \cap R_u$ receive priority and must all be restored, where failing to restore a scheduled request immediately discards the candidate solution. After the operation concludes, requests remaining in R_u form the updated rejected set R_r . For a feasible insertion $\hat{\Omega}$, we define the objective increase:

$$\Delta\mathcal{F}_r^+ := \mathcal{F}(\hat{\Omega}) - \mathcal{F}(\Omega). \quad (81)$$

We implement the following operators:

1. **OR1: Random.** Select requests in random order and apply their minimum-cost, i.e., greedy, feasible insertion. Repeat until no feasible insertions remain.
2. **OR2: Greedy.** Determine the minimum-cost feasible insertion of every request, then apply the insertion with the smallest $\Delta\mathcal{F}_r^+$.
3. **OR3: Regret-2.** For each request, identify its best and second-best insertion costs $\Delta_1\mathcal{F}_r^+$ and $\Delta_2\mathcal{F}_r^+$. Insert the request maximizing $\Delta_2\mathcal{F}_r^+ - \Delta_1\mathcal{F}_r^+$ at its best position, thereby prioritizing requests whose later insertion incurs a substantially higher cost. A request with only one feasible insertion receives immediate priority.
4. **OR4: Longest-Trip.** Select requests in decreasing order of their direct trip length $c_{p^r d^r}$ and apply their minimum-cost feasible insertion. Ties are resolved by the earliest pickup time e_{p^r} . Given our fixed road network, longer trips provide better chances for insertions with comparatively shorter detours.
5. **OR5: Most-Constrained.** Select requests in increasing order of their candidate set size $|\mathcal{C}_{\text{ALNS}}(r, \Omega)|$, ignoring empty candidate sets, and apply minimum-cost feasible insertion. Ties are resolved by the smallest objective increase. Similar to regret-based insertion, this accommodates requests with limited or costly insertion opportunities, in addition to reducing the risk that these requests become infeasible after inserting less constrained ones.

The aforementioned operators align well with our problem characteristics and balance diversification, immediate cost reduction, look-ahead, and insertion flexibility during repair [22]. Since operator effectiveness depends on the problem and neighborhood characteristics, a fundamental advantage of ALNS is its adaptive operator selection, driven by dynamically updated selection weights [12, 31, 32].

5 MODEMS Architecture

This section places our routing and scheduling framework within the full MODEMS architecture. In this work, we implemented the constructive initialization methods, three MILPs, ALNS, and a dynamic simulation layers environment incorporating the rolling-horizon workflow. The remaining service layers and their integration through the

communication middleware are presented as a Proof-of-Concept (PoC); their full-scale development and real-world deployment are deferred to future work.

5.1 Software Components

We propose to employ ROS 2 [36] as the underlying communication middleware, owing to its modularity, asynchronous data exchange, and scalability. Interface messages connect the components depicted in Figure 3: (i) Frontend, (ii) Backend, (iii) Database, (iv) Information-Service, (v) Agent, and (vi) Optimizer. The last two components comprise multiple ROS 2 nodes, one for each available agent or optimizer.

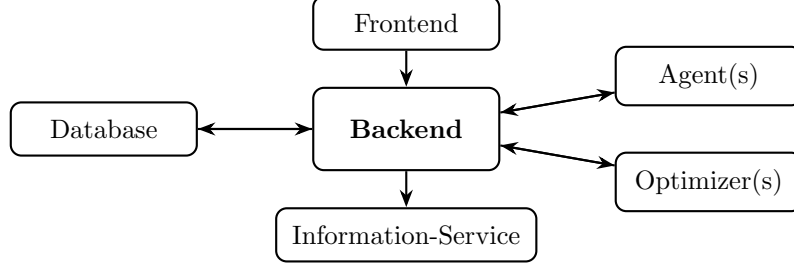


Fig. 3 MODEMS software components and their relationships

The **frontend** gathers and sanitizes user requests, ensuring valid pickup and delivery stations, passenger demand, service time, and earliest pickup time. For sufficient planning and communication time, e^r must lie at least 15 min in the future, where we offer a fixed TW $\omega = 5$ min. Same-day and next-day requests are stored in the **database**, wherein requests outside the active planning horizon remain *pending* until they become relevant. A user may cancel a pending request if its earliest pickup time is at least 60 min in the future. In addition to requests, the database keeps historical records of network data, agent states, and routing solutions. The **information-service** communicates either request acceptance and a confirmed pickup time, or rejection when no service can be offered. To accommodate minor operational delays, the communicated pickup time can marginally precede the planned service start by 1 min to 2 min.

The **backend** coordinates data exchange, compiles routing instances, invokes the optimizers, validates their results, and updates the planned agent journeys. For algorithm evaluation, the implemented *benchmarking* mode sequentially executes MILP1, MILP2, MILP3, and ALNS on a static snapshot, recording the strategy-specific constructive solutions and final optimization results. During rolling-horizon workday simulations, request batches are iteratively revealed to a single optimizer, compiling performance metrics, and using its solution to advance the simulation. The intended *operational* configuration applies the same approach, but executes MILP3 and ALNS concurrently on two dedicated machines, selecting the better feasible solution according to the common objective (49). Finally, fleet **agents** represent the electric shuttles and interact bidirectionally with the backend to exchange journeys, assigned requests, and current operational data.

5.2 Dynamic Agent States

At each planning epoch, we measure time relative to the current decision time. Herein, varying agent parameters v^k , δ^k , and σ^k are compiled from the latest agent state and, if $\Omega_p \neq \emptyset$, the preceding route plan. An agent has one of the following operational states:

- **Out-of-service:** The agent is currently unavailable and is excluded from the set K .
- **Available:** The agent has zero occupancy and no retained requests; its physical position is mapped to v^k with deterministic delay δ^k and SoC σ^k . It is included in K .
- **Charging:** The agent is recharging at a depot mapped to v^k , with remaining charging time δ^k and anticipated SoC σ^k after δ^k has elapsed. If $\delta^k < P$, where P denotes the planning horizon, the agent is included in K as a delayed *available* agent, and excluded otherwise.
- **En route:** The agent is traversing the network, either traveling to a depot or service node. This complex state is explained below and illustrated in Figure 4.
- **Waiting:** The agent is stationary at a service station before service begins; a special case of the *en route* state with zero remaining travel time.

When an agent is *en route*, its next planned node is denoted by n_{i+1} , where δ^k initially represents the remaining travel time to n_{i+1} and its expected arrival SoC σ^k is determined using (46) from the current SoC $\hat{\sigma}^k$ and load $\hat{z}^k$. If n_{i+1} is a final depot, we map the depot to v^k and evaluate $\sigma^k \geq \underline{\sigma}^k + \varepsilon_\sigma$, where ε_σ is a robustness margin that prevents immediate redeployment of an agent with insufficient charge. If the threshold is met, the agent becomes *available* after δ^k ; otherwise, the expected charging duration, approximately 1 h to 2 h for a partial charge or multiple hours for a full charge, is added to δ^k , wherein σ^k denotes the desired SoC after charging. Consequently, we may include or exclude the agent from K , decoupling the charging decision from the routing models.

When an agent is *en route* to a service node with on-board passengers $\hat{z}^k > 0$, it must fulfill all assigned requests until it arrives at a delivery node satisfying zero occupancy after service completion, i.e., until all currently on-board passengers disembark; we propagate the retained route prefix, including all intermediate service nodes, to this delivery. Consequently, we map this physical station to v^k , setting δ^k and σ^k to the anticipated departure time and SoC after service, respectively. Afterwards, if the agent is traveling with an empty load to an assigned pickup p^r , we determine its waiting-time slack $\hat{w}^k = t_{p^r} - \delta^k$ and compare it to the replanning threshold W ; we retain imminent requests with insufficient replanning time $\hat{w}^k \leq W$, proceeding to the first subsequent delivery after which the agent is empty. For $\hat{w}^k > W$, the destination station is mapped to v^k with its anticipated arrival time and SoC, and subsequent request assignments are released for re-optimization. The same comparison applies when the agent is already *waiting* at the pickup station. In all cases, completed, retained, and previously rejected requests are excluded from R , whereas previously accepted requests available for replanning constitute R_s and must remain accepted.

5.3 Rolling-Horizon Workflow

The campus represents a confined environment with average trip times under 5 min and a maximum travel time of approximately 8 min between the furthest service stations.

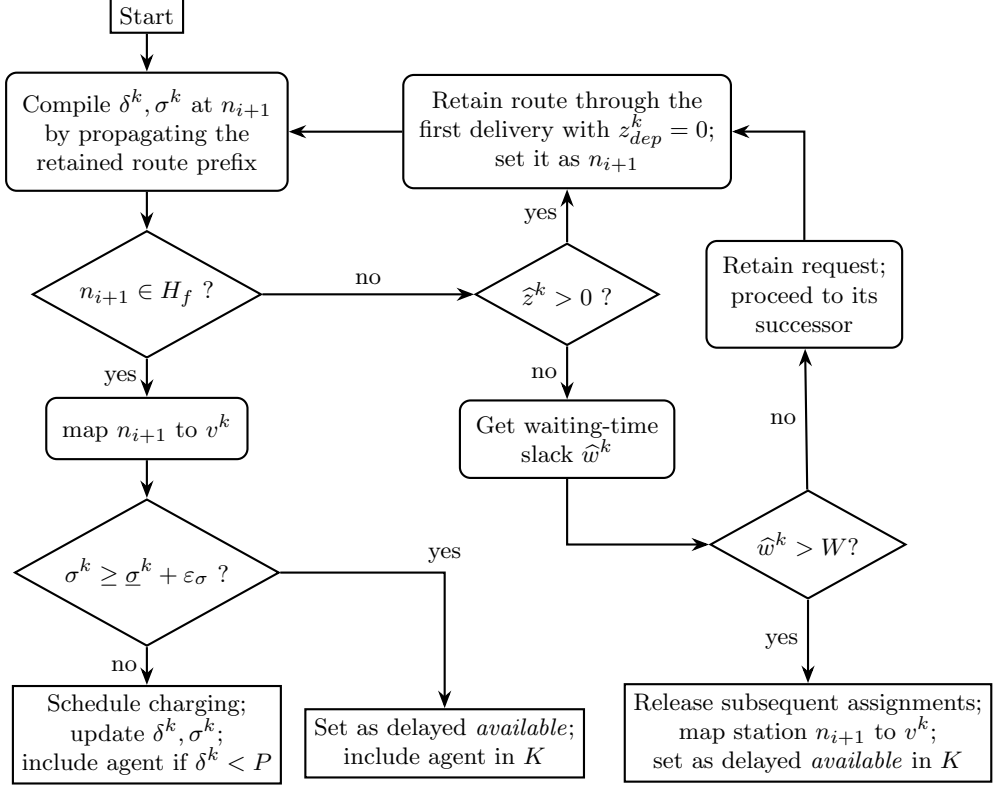


Fig. 4 Workflow for handling the agent *en route* status

Accordingly, we adopt a periodic replanning interval of 10 min, a matching replanning threshold $W = 10$ min, a planning horizon of $P = 60$ min, and a sufficiently large $M = 150$ min to safely deactivate temporal constraints. These provisional values are amenable to change upon real deployment.

The backend initiates a planning epoch every 10 min or whenever a new request is received with an earliest pickup within the planning horizon, $e^r < P$. Previously rejected requests remain rejected and are archived after notifying the user; a later submission is treated as a new request. Pending requests that newly enter the planning horizon constitute R_n , whereas R_s comprises previously accepted requests that have not begun service and remain available for replanning. The current network and compiled agent states yield the static snapshot (R, K, G') , with $R = R_n \cup R_s$. The feasible excerpt of the preceding plan provides the partial plan Ω_p for Algorithms 1 and 2, after which the snapshot is solved with a time limit of 1 min. During workday simulations, the optimizer solution advances the simulated agent states; during operation, it is transmitted to the agents. If the optimizer does not return a feasible update, the preceding feasible plan is retained and all new requests R_n are rejected. The resulting acceptance or rejection decisions for requests in R_n trigger the corresponding user notifications.

5.4 Software Implementation

To promote robustness and facilitate future development, we implemented our codebase entirely in Python and adopted existing software packages. Specifically, we used Pyomo [37] in conjunction with the academic license of Gurobi [38] for developing and solving the three MILP models, respectively; for ALNS, we used the similarly named ALNS package [39]. Moreover, the codebase includes independent classes for the constructive heuristics, problem components, unified solution representation, and a dynamic simulation environment to ensure a comprehensive evaluation of the intended MODEMS architecture. The environment generates deterministic snapshots, mimics request submission streams, advances agent journeys, prepares problem instances, invokes routing strategies, and records generated solutions and performance metrics. Accordingly, it constitutes the basis for static comparative studies and dynamic rolling-horizon experiments in Section 6. In accordance with *FAIR* principles [40], we have documented and uploaded our codebase to <https://github.com/memam-hub/modems>, including the benchmarking scripts used to generate the achieved results.

6 Numerical Experiments

In this section, we conduct three complementary numerical studies: (i) an ablation study to quantify the computational benefits of our proposed feasibility-driven insertion procedure; (ii) a static comparison of MILP1, MILP2, MILP3, and ALNS under identical scenarios; and (iii) a dynamic workday study to analyze the rolling-horizon performance of MILP3 and ALNS under different routing objectives and fleet-dispatching strategies.

Table 4 Parameters used in all numerical experiments

Agent		Model		Roulette Wheel		Simulated Annealing	
σ	0.25	ϵ	0.01	<i>best</i> score	5	start temperature	1000
α	6.65×10^{-3}	ζ	2.0	<i>better</i> score	2	end temperature	0.1
β	0.15×10^{-3}	η	100	<i>accepted</i> score	1	cooling rate	0.995
		ω	5.0	<i>rejected</i> score	0.5		
		ρ	2.5	decay rate	0.8		

Throughout the experiments, we characterize each scenario by its size and by the spatial and temporal distributions of its generated requests. *Small*, *medium*, and *large* scenarios comprise one, two, and three agents serving $|R| \in [4, 8]$, $[5, 12]$, and $[10, 20]$ requests, respectively. *Random* scenarios draw request origins and destinations uniformly across the service network, *clustered* scenarios concentrate them around sampled activity centers, and *mixed* scenarios combine both patterns. *Uniform* timing distributes earliest pickup times uniformly throughout the planning horizon, whereas *peaks* timing concentrates half of the workday demand around two high-demand periods. The individual studies customize and extend these characteristics according to their respective objectives. To enhance comparability, all experiments adopt identical parameter values, which are listed in Table 4.

6.1 Feasible-Insertion Ablation Study

The ablation study comprises four incremental variants, each identifying the feasible insertions and corresponding objective values for an unassigned request into a feasible partial plan. All variants enforce identical feasibility conditions and differ exclusively in their candidate-identification, state-propagation, and pruning mechanisms.

1. **V0: Naive.** Enumerate every precedence-feasible insertion pair (a, b) satisfying $0 \leq a \leq b < m_k$, then sequentially attempt each pair, propagating the modified journey from position a to the final depot and retaining candidates without feasibility violations. No information is reused across insertion pairs.
2. **V1: Capacity Pre-filtering.** Extend V0 with a computationally inexpensive check against the original route-load profile. An insertion pair is discarded before construction whenever the additional demand q^u proves it capacity-infeasible.
3. **V2: Incremental Prefix Reuse.** Extend V1 by constructing the prefix once for every pickup position a and incrementally extending it as the delivery position b advances. Every surviving pair still performs unconditional state propagation from b to the final depot; neither monotone pruning nor lazy suffix propagation is applied.
4. **V3: Proposed Method.** Apply Algorithm 3 with incremental prefix reuse, monotone pruning, and lazy suffix propagation, terminating temporal propagation once the insertion-induced shift is absorbed.

Each measurement starts from a partial solution wherein all but three requests have been greedily inserted; the three held-out requests act as independent insertion probes. We consider scenarios with 10 to 60 requests, $|K| \in \{1, 2, 3\}$, three spatial distributions, two temporal profiles, and three repetitions. This yields 432 partial solutions and 1296 probes per corpus, while every probe is measured five times for each variant to stabilize measurement times. The normal and stress corpora use identical seeds and differ only in their initial SoC ranges, $\sigma \in [0.8, 1.0]$ and $\sigma \in [0.5, 0.7]$, respectively.

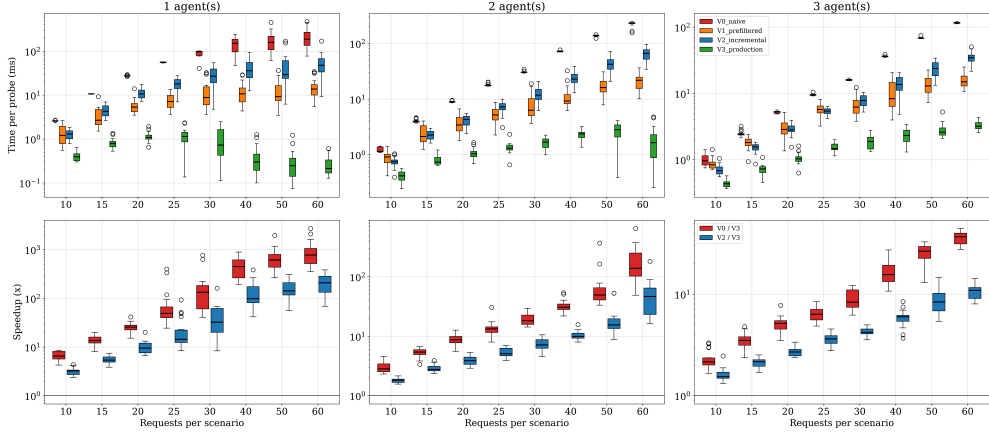


Fig. 5 Feasible-insertion runtimes (in milliseconds) and speedups for the normal initial SoC range $\sigma \in [0.8, 1.0]$, grouped by number of agents and requests.

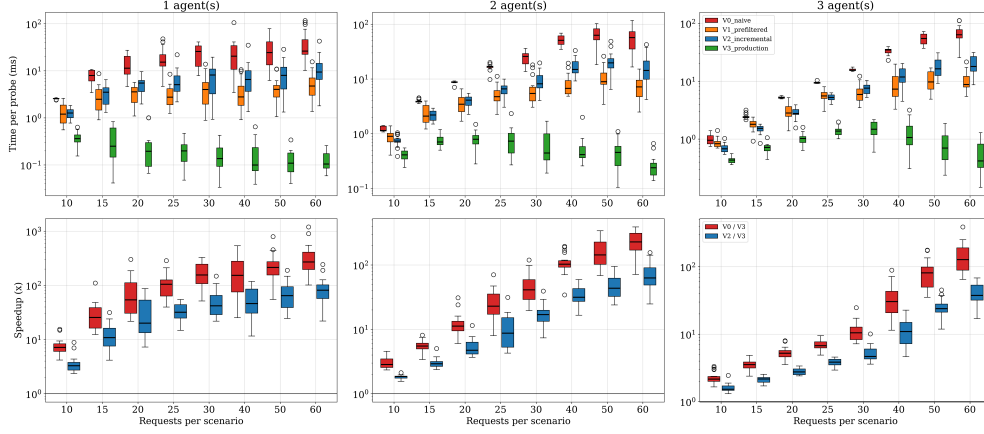


Fig. 6 Feasible-insertion runtimes (in milliseconds) and speedups for the restricted initial SoC range $\sigma \in [0.5, 0.7]$, grouped by number of agents and requests.

Figures 5 and 6 demonstrate that capacity pre-filtering substantially reduces the naive enumeration cost, while incremental prefix reuse independently provides limited benefit. In particular, the bookkeeping required by V2 is not rectified when every surviving pair still undergoes unconditional suffix propagation, yielding runtimes often slower than V1. Conversely, V3 consistently maintains a runtime of 5 milliseconds or less, achieving speedups orders of magnitude higher than its competitors. Moreover, we observe that V3 converges to a runtime of under one millisecond for one agent within the normal SoC range and across all agents within the restricted SoC range. This denotes that the agent journey is at maximum capacity; any subsequent insertion attempt often violates feasibility constraints, thereby accelerating the pruning process.

Bucket	V0/V1		V1/V2		V2/V3		V0/V3	
	normal	stress	normal	stress	normal	stress	normal	stress
[0, 10)	1.31	1.32	1.22	1.22	1.71	1.71	2.67	2.69
[10, 20)	2.01	2.02	1.04	1.04	3.08	3.79	5.96	7.61
[20, 30)	3.84	4.34	0.74	0.73	5.68	19.58	14.53	60.57
[30, 40)	6.09	6.76	0.52	0.54	9.73	47.69	28.01	157.75
[40, 50)	8.74	8.17	0.42	0.45	23.72	70.32	96.13	229.64
[50, 60)	11.40	13.24	0.36	0.38	69.24	132.81	257.58	593.50
[60, 70)	14.40	15.55	0.33	0.30	161.82	189.61	677.69	898.51
[70, 80)	14.16	—	0.31	—	176.75	—	677.39	—
[80, 90)	19.49	—	0.24	—	196.32	—	885.37	—
[90, 100)	22.33	—	0.21	—	271.36	—	1468.33	—

Table 5 Feasible-insertion speedups by effective route length, i.e., mean number of route nodes per agent. Each value represents the mean within bucket $[a, b)$, where $V_i/V_j > 1$ favors V_j . The first three ratios isolate successive optimization steps and V0/V3 denotes the overall speedup.

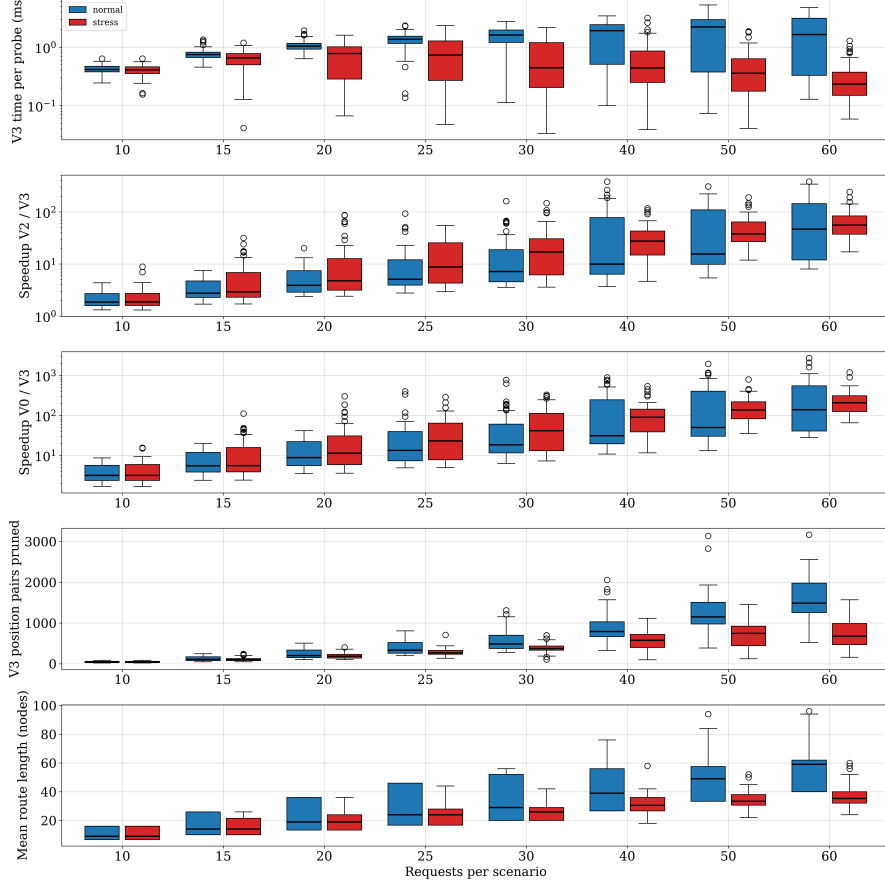


Fig. 7 Comparison of the normal and stress corpora by the number of requests. Panels depict $V3$ runtime, incremental and overall speedups, pruned position pairs, and effective route length.

Figure 7 highlights that the restricted SoC budget typically yields shorter agent routes for otherwise identical instances. Consequently, we base our analysis in Table 5 on the effective route length, i.e., the mean number of route nodes per agent, rather than the number of requests. Compared to $V0$, the effectiveness of $V1$ is emphasized as route length increases, whereas $V2$ loses its advantage to $V1$ beyond the smallest route-length buckets. In contrast, the performance and speedups of $V3$ consistently improve across all route lengths. For each pruned insertion pair, we spare the subsequent propagation costs, where the monotone energy bound is particularly effective when an agent journey reaches its maximum capacity, e.g., in the restricted SoC range. Overall, our proposed procedure combines reliable feasibility screening and efficient performance for ALNS construction and repair processes.

6.2 Single Solver Evaluation

Next, we evaluate each routing strategy using 54 static instances, with *small*, *medium*, and *large* scenarios, three spatial demand distributions, two timing profiles, and three repetitions. Earliest pickup times lie within a 15 min to 60 min horizon with fixed 5 min pickup windows, resembling the rolling-horizon temporal footprint, and agents receive initial SoC values $\sigma^k \in [0.8, 1.0]$. As explained in Section 4.4, each strategy is initialized using its associated constructive method. The MILPs are warm-started and solved by Gurobi with a 60s time limit, whereas ALNS terminates after 1000 consecutive iterations without incumbent improvements.

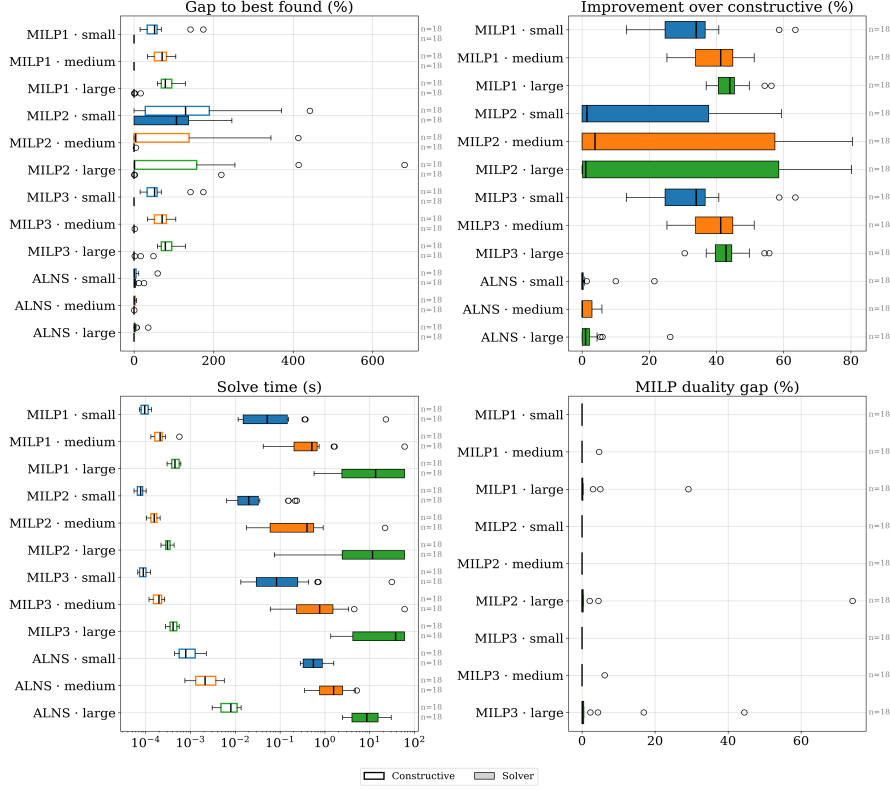


Fig. 8 Static solver comparison by strategy and size. Panels depict the constructive (hollow) and solver (solid) gap to the best found solution for each instance, improvement over the associated constructive solution, solution time (in seconds), and final MILP duality gap. Grouped by scenario size, each box represents 18 instances. All constructive solutions finish in under 20 milliseconds.

Figure 8 summarizes the performance of our four strategies and their associated baselines. MILP1 and MILP3 generally remain close to the best reported objective; MILP2 exhibits a significantly wider distribution, especially for small instances where hard TWs often result in request rejection with high rejection penalties. Herein, the figure clarifies that the large and highly variable improvements of MILP2 are attributed

to its inadequate constructive baseline, rather than to an exceptional performance. Similarly, MILP1 and MILP3 report considerable, albeit insignificant, improvements to their constructive solutions. Conversely, the smaller ALNS improvements emphasize that its feasibility-driven construction already provides a competitive incumbent.

On the one hand, solution times for all three MILPs heavily rely on the scenario size, increasing by orders of magnitude. Our efficient formulations typically yielded optimal solution within the time budget, while the corresponding duality-gap panel highlights isolated large or challenging instances that terminated without an optimality certificate. On the other hand, ALNS provides a narrower and more predictable runtime distribution while maintaining small gaps to the best achieved solutions. Therefore, the aggregate results support the complementary use of the exact and metaheuristic strategies: when tractable, MILPs yield strong solutions backed by optimality certificates, whereas ALNS maintains dependable feasibility and solution quality as the search space expands.

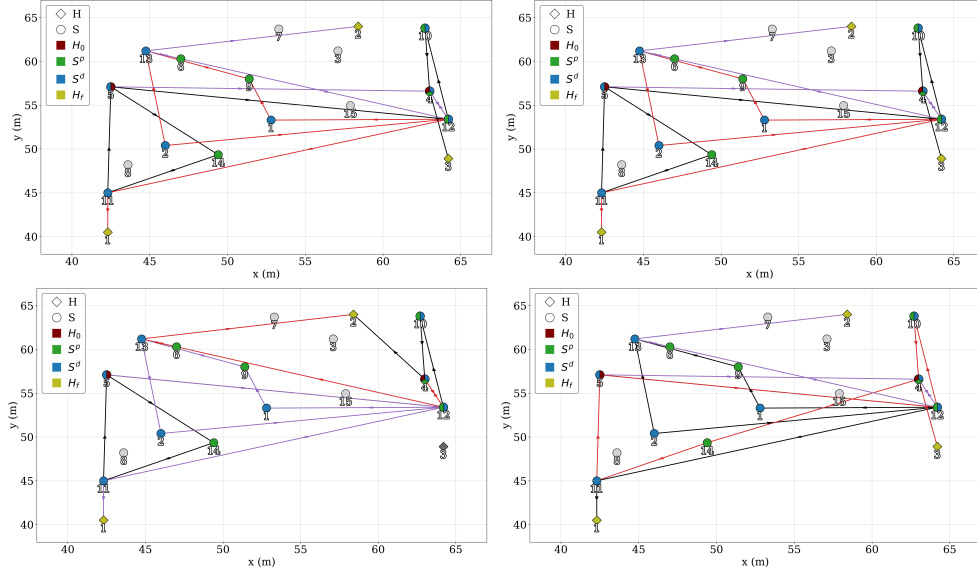


Fig. 9 Solved routes for SC0_LMP50_70, obtained by MILP1, MILP2, MILP3, and ALNS from top left to bottom right. Colors distinguish the three agent journeys, while node markers represent their physical type and color denote their multiple virtual roles.

To illustrate the achieved routing plans, we additionally consider the representative instance SC0_LMP50_70, comprising three agents, twelve requests, *mixed* spatial demand, pickup times within *peaks*, and restricted initial SoC $\sigma^k \in [0.5, 0.7]$. Herein, all strategies achieve an identical objective of ≈ 60.382 , where Figures 9 and 10 demonstrate that substantially different route assignments can satisfy the same precedence, capacity, ride-time, and energy requirements. This single instance is illustrative rather than representative of the aggregate comparison; it emphasizes the scheduling flexibility under an identical objective function, even within exact methods.

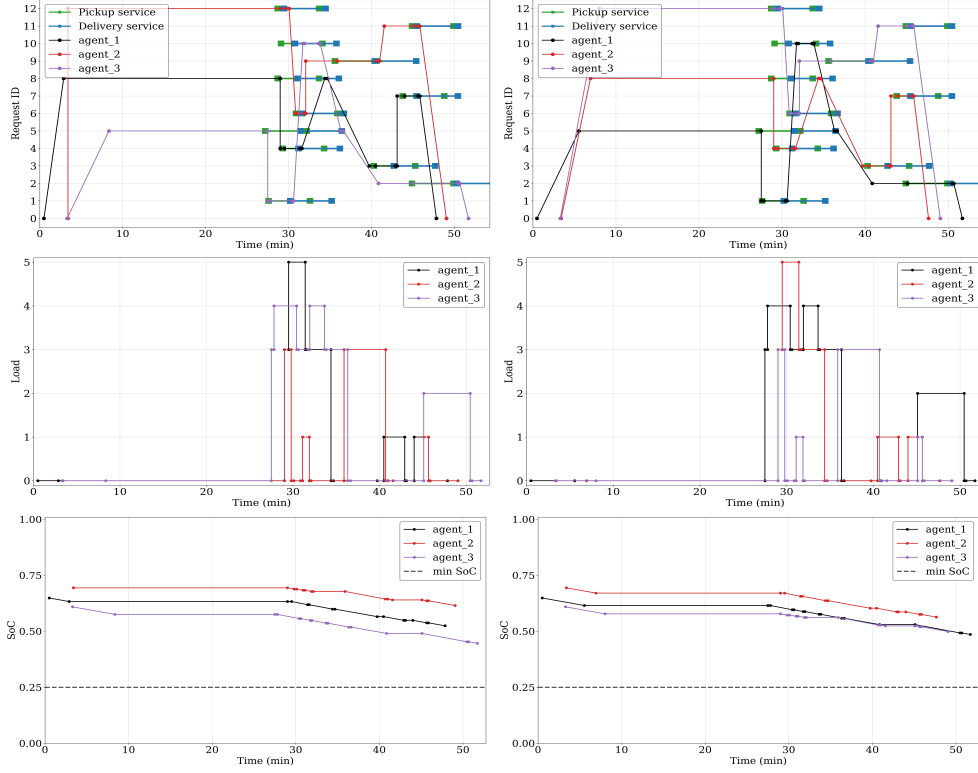


Fig. 10 Timing, load, and SoC profiles for scenario SC0_LMP50_70, as solved by (left) MILP3 and (right) ALNS.

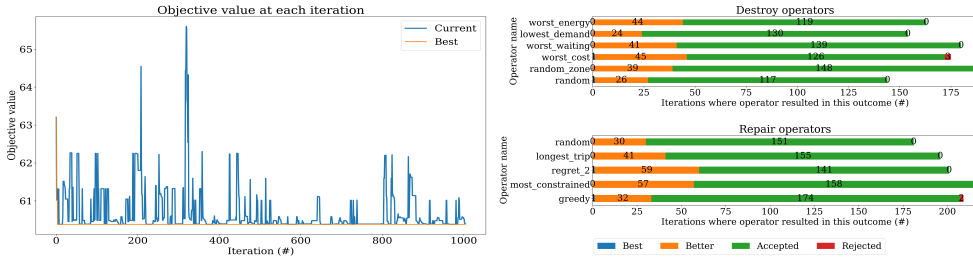


Fig. 11 Aggregated ALNS metrics for SC0_LMP50_70: (left) incumbent and current objective value progression; (right) destroy and repair operator outcomes.

For ALNS, Figure 11 shows that the constructive incumbent is improved within the first iterations, after which the search explores inferior current solutions while retaining the best solution. All destroy and repair operators contribute accepted moves, whereas a constrained number of applications establish a new incumbent. This behavior aligns with our intended design to balance diversification and intensification using adaptive operator selection.

6.3 Dynamic Workday Evaluation

Finally, we evaluate the rolling-horizon framework over twelve independently generated 480 min workdays, operating three fleet agents with initial $\sigma^k = 1.0$. Requests are submitted at base rates of either five or eight requests per hour, follow *uniform* or *peaks* temporal profiles, and include an additional 30 min demand surge. Every workday is simulated using MILP3 and ALNS, each solved with closed and open routing objectives, wherein agents start at *normal* or *staggered* times, yielding a total of 96 workday executions. *Normal* deployment makes the complete fleet available at the beginning of the day, whereas *staggered* deployment releases successive agents at 96 min intervals.

In our analysis, we consider request acceptance, mean pickup delay, mean excess ride time, and the fleet travel time and energy consumption per served request. The latter two metrics normalize resource consumption by the achieved service volume; therefore, they represent the operational effort per accepted request.

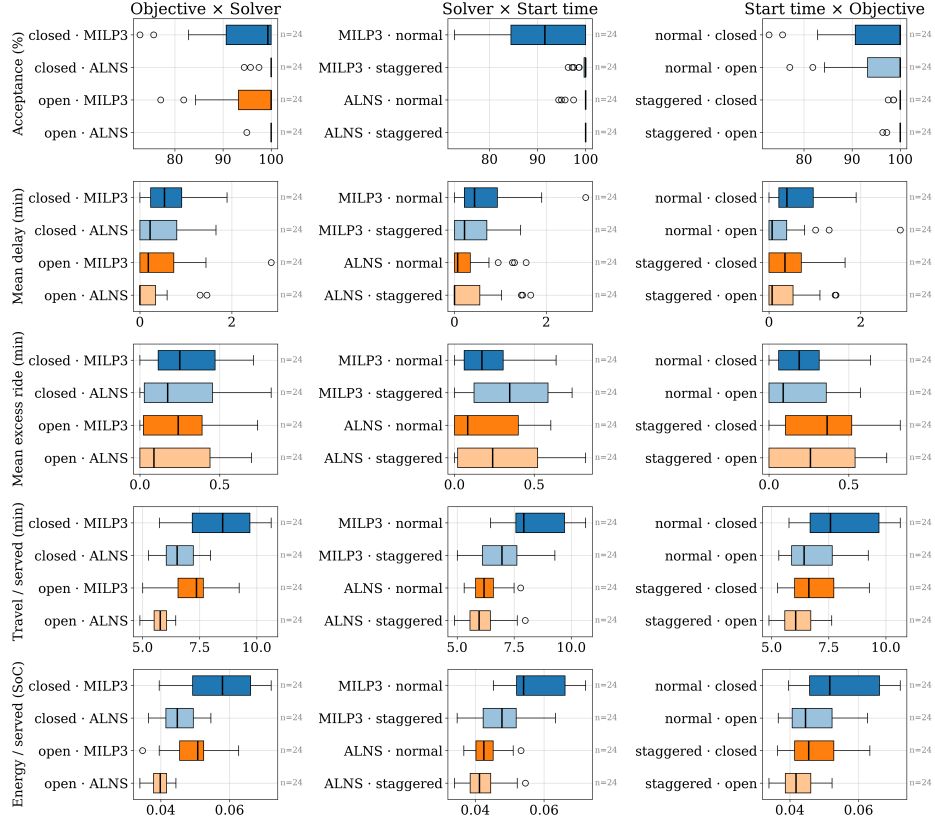


Fig. 12 Workday outcome distributions across routing objective, solver, and fleet start time. Rows represent the five evaluation metrics, while columns group the interactions between two investigated decisions. Each box summarizes 24 workdays.

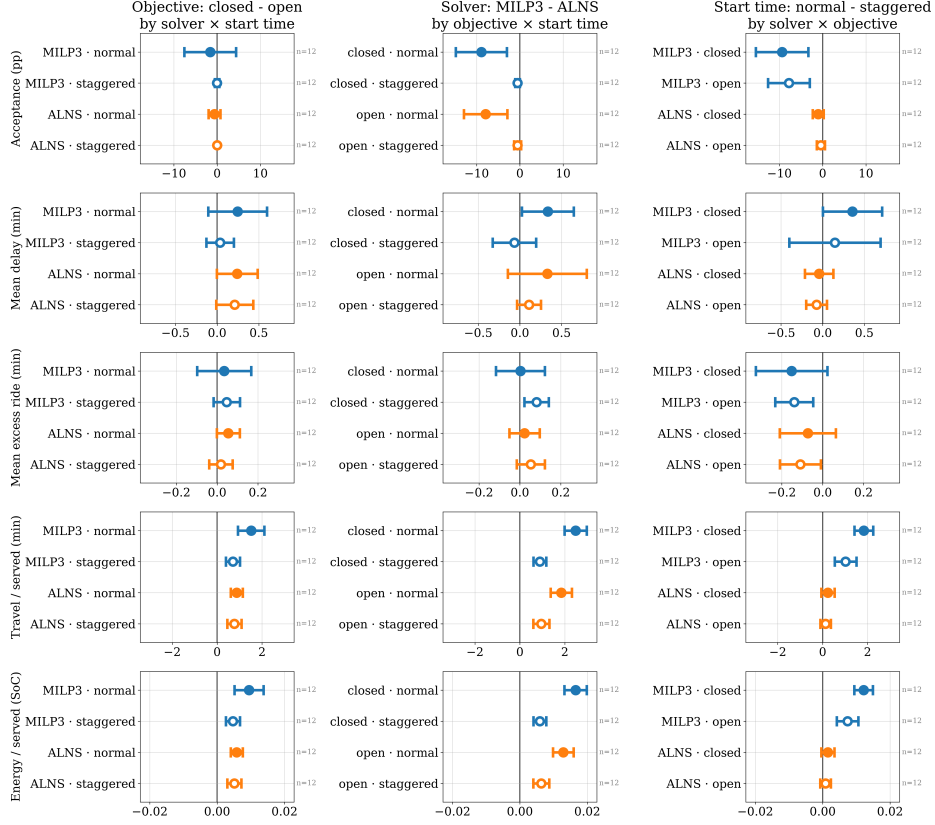


Fig. 13 Paired comparisons with 95% confidence intervals. Differences are computed on identical request demands as closed minus open (objective), MILP3 minus ALNS (solver), and normal minus staggered (deployment strategy). Positive values favor the first alternative only for acceptance, while the second alternative performs better across all remaining metrics.

Figure 12 shows that, compared to MILP3, ALNS achieves near-complete acceptance across evaluated configurations, while requiring significantly less time and energy per served request. The paired comparisons in Figure 13 confirm these findings, with MILP3 consistently consuming more resources on identical workdays.

During actual deployment, MILP3 frequently encounters difficulties in improving incumbents within the strict time budget; combining this with its weak constructive heuristic, the effect accumulates over the workday, yielding an overall worse performance. Furthermore, MILP3 is more sensitive to the fleet deployment strategy: normal deployment produces a wider acceptance distribution and higher resource consumption, whereas staggered starts considerably narrows these differences. We attribute this to the cumulative objective function that indiscriminately utilizes all available agents, often resulting in service outages, wherein all agents are simultaneously charging. During these periods all received requests are rejected. However, staggered deployment de-synchronizes agent depletion for MILP3, thereby preserving later fleet availability. On the other hand, adaptive ALNS destroy operators, such as *lowest-demand*, mitigate

the impact of inefficient request assignment among agents, substituting for a more complex, non-myopic, or lexicographic objective function. In all cases, the open objective reduces travel and energy as expected, indirectly improving the acceptance rate; nevertheless, its contributions to delay and excess ride time remain inconclusive.

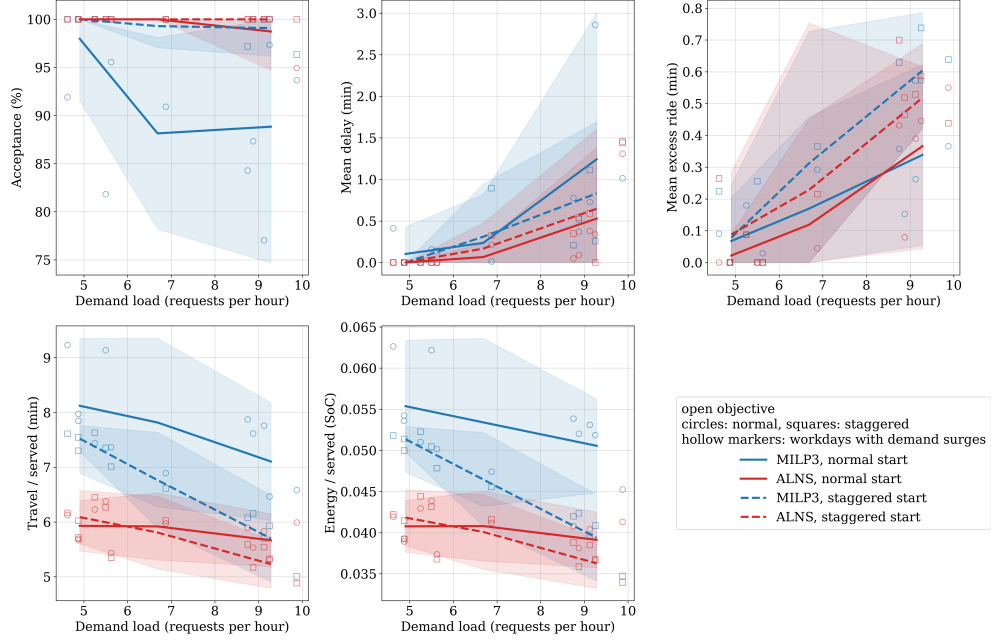


Fig. 14 Open-objective workday outcomes against realized demand. Each marker summarizes one complete workday. Lines connect bin means, and shaded regions depict pointwise 95% confidence intervals obtained from four workdays per demand bin.

Figure 14 further demonstrates that increasing demand primarily challenges MILP3 under normal fleet deployment, whereas ALNS and staggered starts maintain high acceptance. Pickup delays and excess ride times typically increase with demand, albeit with considerable workday-level variation. Conversely, travel and energy per served request decrease, indicating improved pooling and fleet utilization as more combinable requests become available. Since every bin only comprises four workdays, the confidence bands remain wide and shall be interpreted descriptively; the more informative analysis of experimental decisions is provided by the paired comparisons in Figure 13.

Figures 15 and 16 illustrate how aggregate effects emerge from individual operational trajectories. Under normal deployment, similar depletion patterns for MILP3 result in overlapping charging periods and, consequently, zero-acceptance periods; this is mitigated by staggered deployment. Furthermore, MILP3 generates longer outage periods at the workday with the largest number of requests compared to the workday with the largest passenger volume. This aligns with the MILP3 solving complexity: more requests induce a higher number of constraints, subsequently increasing the runtime

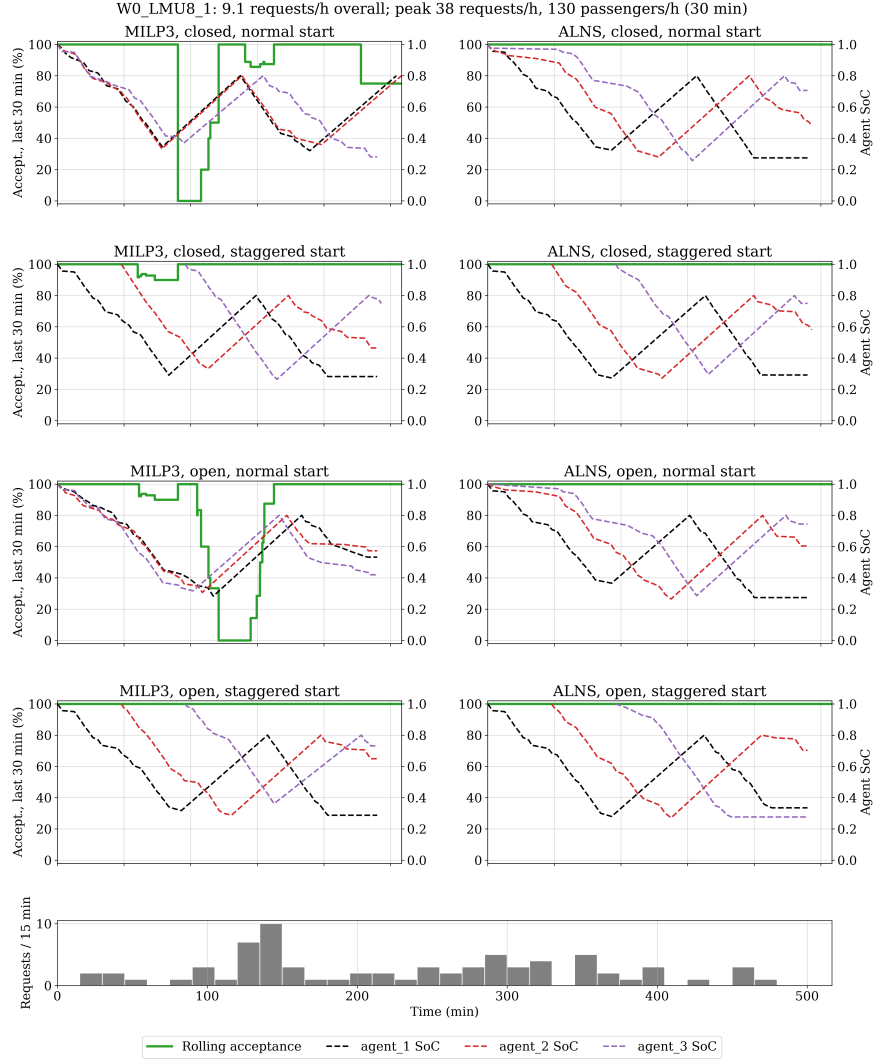


Fig. 15 Rolling 30 min acceptance, agent SoC, and submitted requests for the workday with the largest number of requests. Panels offer the full decision matrix: rows combine closed and open objectives with normal and staggered starts, while columns compare MILP3 (left) and ALNS (right).

and yielding worse incumbents, whereas the passenger volume remains insignificant with respect to solution time. In contrast, ALNS produces consistently robust results for the workday with the highest number of requests, whereas it struggles with the workday with the largest passenger volume. Herein, only staggered dispatching was able to accommodate the 30 min demand surge of 132 passengers per hour without dropping below 100% acceptance rate.

Tables 6–7 depict the open-objective, staggered-start request log for the workday with the highest number of requests. Both solvers service all 73 requests, although

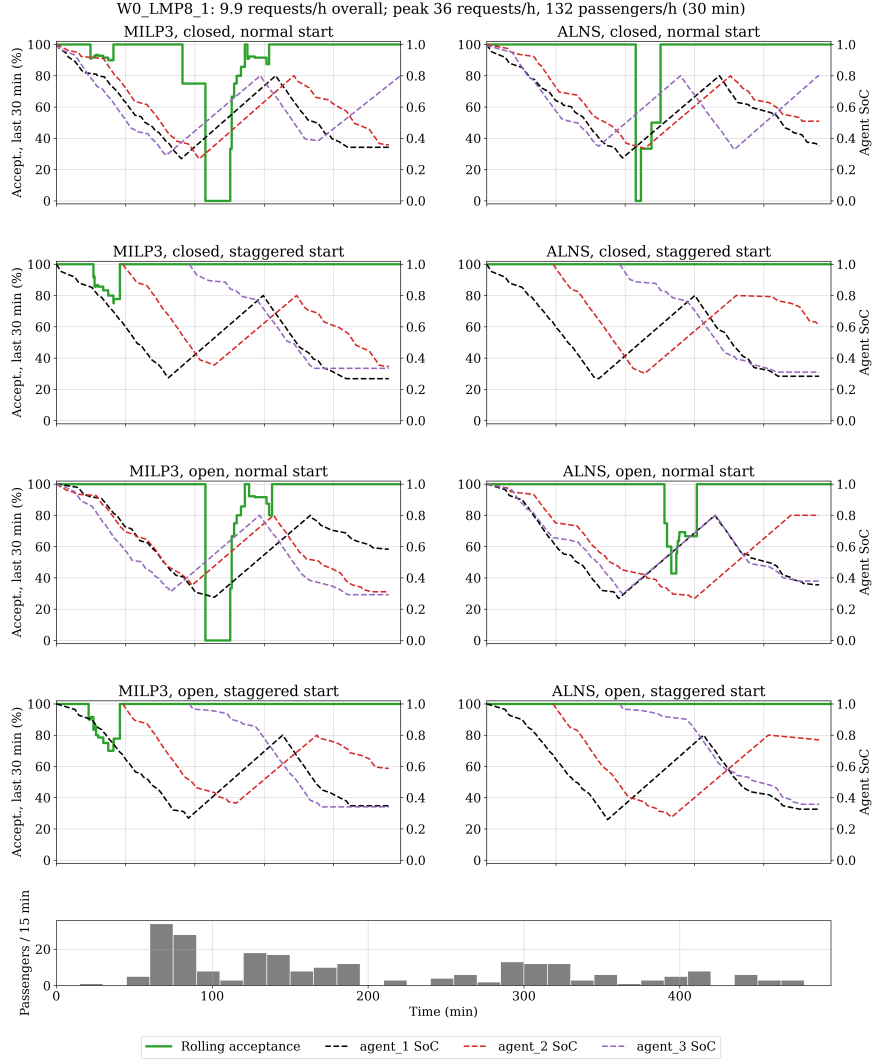


Fig. 16 Rolling 30 min acceptance, agent SoC, and submitted passenger demand for the workday with the largest passenger volume. Panels offer the full decision matrix: rows combine closed and open objectives with normal and staggered starts, while columns compare MILP3 (left) and ALNS (right).

ALNS achieves lower mean pickup delay and marginally lower excess ride time. Tables 8–9 present the corresponding log for the passenger volume peak. Here, MILP3 services 76 of 79 requests, whereas ALNS services all requests with slightly lower mean delay and excess ride time. The request-level results demonstrate that similar daily aggregates may arise from substantially different assignments and schedules, and that service outages remain an artifact of unplanned high-demand periods.

Key	Submission time	Earliest pickup	Accepted?		Pickup time		Delivery time		Delay (min)		Excess ride (min)	
			MILP3	ALNS	MILP3	ALNS	MILP3	ALNS	MILP3	ALNS	MILP3	ALNS
RQ_05c5	08:01	08:24	✓	✓	08:24	08:24	08:26	08:26	0.0	0.0	0.0	0.0
RQ_fd96	08:04	08:35	✓	✓	08:36	08:36	08:40	08:40	0.0	0.0	0.0	0.0
RQ_83f0	08:07	08:23	✓	✓	08:29	08:29	08:32	08:32	0.6	0.6	0.0	0.0
RQ_6782	08:17	08:44	✓	✓	08:44	08:44	08:47	08:47	0.0	0.0	0.0	0.0
RQ_30e2	08:22	08:58	✓	✓	08:58	08:58	09:01	09:01	0.0	0.0	0.0	0.0
RQ_b695	08:49	09:27	✓	✓	09:27	09:27	09:31	09:31	0.0	0.0	0.0	0.0
RQ_1546	08:59	09:42	✓	✓	09:42	09:42	09:44	09:44	0.0	0.0	0.0	0.0
RQ_f7ae	09:03	09:31	✓	✓	09:31	09:31	09:35	09:35	0.0	0.0	0.0	0.0
RQ_8c7f	09:10	09:43	✓	✓	09:43	09:43	09:47	09:47	0.0	0.0	0.0	0.0
RQ_c80d	09:20	10:02	✓	✓	10:02	10:02	10:05	10:05	0.0	0.0	0.0	0.0
RQ_3687	09:22	09:49	✓	✓	09:49	09:49	09:55	09:55	0.0	0.0	0.0	0.0
RQ_dadd	09:26	10:10	✓	✓	10:16	10:10	10:18	10:12	1.4	0.0	0.0	0.0
RQ_d65b	09:28	10:12	✓	✓	10:23	10:16	10:27	10:20	5.7	0.0	0.0	0.0
RQ_8c8e	09:37	10:21	✓	✓	10:28	10:28	10:33	10:33	1.4	1.5	1.8	1.8
RQ_3e67	09:41	10:07	✓	✓	10:07	10:07	10:09	10:09	0.0	0.0	0.0	0.0
RQ_69b5	09:42	10:14	✓	✓	10:18	10:14	10:20	10:16	0.0	0.0	0.0	0.0
RQ_8366	09:44	10:07	✓	✓	10:07	10:07	10:13	10:13	0.0	0.0	0.0	0.0
RQ_0524	09:45	10:29	✓	✓	10:34	10:32	10:38	10:36	0.0	0.0	0.4	0.2
RQ_cff4	09:48	10:11	✓	✓	10:12	10:18	10:15	10:21	0.0	2.6	0.0	0.0
RQ_d50a	09:50	10:16	✓	✓	10:33	10:33	10:38	10:38	12.2	5.1	0.0	0.0
RQ_7639	09:56	10:17	✓	✓	10:18	10:21	10:20	10:23	0.0	0.0	0.0	0.0
RQ_4f38	09:58	10:29	✓	✓	10:40	10:35	10:45	10:40	5.7	0.4	2.0	2.9
RQ_56af	09:58	10:27	✓	✓	10:29	10:29	10:31	10:31	0.0	0.0	0.0	0.0
RQ_f6cf	09:59	10:17	✓	✓	10:21	10:22	10:26	10:26	0.0	0.0	0.0	0.0
RQ_bcf4	10:00	10:27	✓	✓	10:28	10:28	10:31	10:31	0.0	0.0	0.0	0.0
RQ_3b39	10:02	10:23	✓	✓	10:31	10:24	10:32	10:25	2.6	0.0	0.0	0.0
RQ_294f	10:05	10:30	✓	✓	10:44	10:36	10:48	10:41	9.0	0.8	0.0	0.4
RQ_51f5	10:06	10:30	✓	✓	10:34	10:31	10:38	10:36	0.0	0.0	0.0	0.8
RQ_bca2	10:06	10:32	✓	✓	10:40	10:40	10:49	10:45	3.2	3.3	4.1	0.0
RQ_bbbc	10:13	10:30	✓	✓	10:38	10:45	10:42	10:47	3.7	9.9	0.5	0.0
RQ_59d3	10:26	11:03	✓	✓	11:03	11:03	11:06	11:06	0.0	0.0	0.0	0.0
RQ_a92e	10:28	10:59	✓	✓	10:59	10:59	11:05	11:05	0.0	0.0	0.0	0.0
RQ_d3fa	11:01	11:28	✓	✓	11:28	11:28	11:31	11:31	0.0	0.0	0.0	0.0
RQ_134e	11:03	11:21	✓	✓	11:21	11:21	11:24	11:24	0.0	0.0	0.0	0.0
RQ_5556	11:14	11:34	✓	✓	11:34	11:34	11:38	11:38	0.0	0.0	0.0	0.0
RQ_631e	11:23	11:40	✓	✓	11:40	11:40	11:46	11:46	0.0	0.0	0.0	0.0
RQ_a6e9	11:30	12:11	✓	✓	12:11	12:11	12:17	12:17	0.0	0.0	0.0	0.0
RQ_a004	11:30	11:48	✓	✓	11:48	11:48	11:51	11:51	0.0	0.0	0.0	0.0

Table 6 Request-level rolling-horizon results for a simulated workday (WO0.LMUS8.1: open objective, staggered start; cont. 1/2). Pickup and delivery times represent the realized service-start times. Delay denotes pickup tardiness, while excess ride time is measured relative to the direct trip duration. Accepted and rejected requests are denoted by ✓ and ×, respectively.

Key	Submission time	Earliest pickup	Accepted?		Pickup time		Delivery time		Delay (min)		Excess ride (min)	
			MILP3	ALNS	MILP3	ALNS	MILP3	ALNS	MILP3	ALNS	MILP3	ALNS
RQ_7dee	11:35	12:20	✓	✓	12:20	12:20	12:22	12:22	0.0	0.0	0.0	0.0
RQ_14d1	11:47	12:14	✓	✓	12:18	12:14	12:24	12:20	0.0	0.0	0.0	0.0
RQ_77be	11:49	12:05	✓	✓	12:05	12:05	12:08	12:08	0.0	0.0	0.0	0.0
RQ_fc99	12:02	12:31	✓	✓	12:31	12:31	12:36	12:35	0.0	0.0	1.0	0.0
RQ_f9b7	12:03	12:36	✓	✓	12:43	12:37	12:53	12:45	2.8	0.0	5.8	3.3
RQ_8677	12:11	12:28	✓	✓	12:28	12:29	12:32	12:31	0.0	0.0	1.0	0.0
RQ_5558	12:12	12:54	✓	✓	12:58	12:55	13:00	12:56	0.0	0.0	0.0	0.0
RQ_ee88	12:15	12:48	✓	✓	12:50	12:48	12:55	12:51	0.0	0.0	2.2	0.0
RQ_2550	12:18	12:33	✓	✓	12:40	12:34	12:46	12:40	2.1	0.0	1.2	1.2
RQ_a43d	12:23	13:01	✓	✓	13:08	13:06	13:13	13:16	1.5	0.0	1.3	5.5
RQ_0ff3	12:31	12:57	✓	✓	13:03	13:00	13:05	13:01	1.1	0.0	0.0	0.0
RQ_af9f	12:37	13:05	✓	✓	13:15	13:17	13:25	13:27	5.0	7.1	5.7	5.7
RQ_84e1	12:39	12:56	✓	✓	13:19	13:03	13:22	13:06	6.1	1.9	0.0	0.3
RQ_e4bc	12:39	13:00	✓	✓	13:12	13:14	13:19	13:21	6.8	8.9	3.3	3.3
RQ_4238	12:42	13:03	✓	✓	13:06	13:09	13:10	13:12	0.0	1.0	0.9	0.0
RQ_e3c6	12:50	13:26	✓	✓	13:33	13:28	13:36	13:30	2.0	0.0	0.0	0.0
RQ_1fd5	12:59	13:26	✓	✓	13:27	13:29	13:33	13:34	0.0	0.0	3.3	1.4
RQ_b47b	13:04	13:46	✓	✓	13:46	13:46	13:48	13:48	0.0	0.0	0.0	0.0
RQ_f513	13:04	13:30	✓	✓	13:39	13:34	13:43	13:38	3.9	0.0	0.0	0.0
RQ_b65a	13:08	13:26	✓	✓	13:26	13:30	13:30	13:37	0.0	0.0	0.3	1.9
RQ_8854	13:09	13:49	✓	✓	13:57	13:50	14:00	13:53	2.5	0.0	0.0	0.0
RQ_3935	13:11	13:47	✓	✓	13:50	13:47	13:52	13:49	0.0	0.0	0.0	0.0
RQ_5d61	13:25	14:03	✓	✓	14:03	14:03	14:06	14:06	0.0	0.0	0.2	0.0
RQ_d2a6	13:33	13:54	✓	✓	13:55	13:54	13:56	13:54	0.0	0.0	0.0	0.0
RQ_e940	13:37	14:18	✓	✓	14:18	14:18	14:19	14:19	0.0	0.0	0.0	0.0
RQ_5f70	13:39	13:59	✓	✓	14:06	13:59	14:09	14:02	1.7	0.0	0.2	0.0
RQ_64af	13:45	14:14	✓	✓	14:14	14:14	14:24	14:24	0.0	0.0	4.6	5.0
RQ_ef1c	13:49	14:32	✓	✓	14:37	14:35	14:42	14:40	0.2	0.0	0.0	0.0
RQ_5c00	14:03	14:30	✓	✓	14:30	14:30	14:33	14:33	0.0	0.0	0.0	0.0
RQ_df0f	14:03	14:40	✓	✓	14:40	14:44	14:46	14:50	0.0	0.0	0.0	0.0
RQ_4330	14:24	15:03	✓	✓	15:03	15:03	15:06	15:06	0.0	0.0	0.0	0.0
RQ_8541	15:10	15:40	✓	✓	15:40	15:40	15:47	15:50	0.0	0.0	2.0	4.8
RQ_084d	15:18	15:41	✓	✓	15:41	15:41	15:43	15:43	0.0	0.0	0.0	0.0
RQ_87f7	15:20	15:45	✓	✓	15:45	15:45	15:48	15:49	0.0	0.0	0.0	0.0
RQ_80f1	15:36	15:59	✓	✓	15:59	15:59	16:02	16:02	0.0	0.0	0.0	0.0

Table 7 Request-level rolling-horizon results for a simulated workday (WO0.LMUS8.1: open objective, staggered start; cont. 2/2). Pickup and delivery times represent the realized service-start times. Delay denotes pickup tardiness, while excess ride time is measured relative to the direct trip duration. Accepted and rejected requests are denoted by ✓ and ×, respectively.

Key	Submission time	Earliest pickup	Accepted?		Pickup time		Delivery time		Delay (min)		Excess ride (min)	
			MILP3	ALNS	MILP3	ALNS	MILP3	ALNS	MILP3	ALNS	MILP3	ALNS
RQ_fa3a	08:02	08:21	✓	✓	08:21	08:21	08:27	08:27	0.0	0.0	0.0	0.0
RQ_0477	08:09	08:47	✓	✓	08:47	08:47	08:51	08:51	0.0	0.0	0.0	0.0
RQ_3dde	08:12	08:56	✓	✓	08:56	08:56	08:58	08:58	0.0	0.0	0.0	0.0
RQ_6498	08:23	09:03	✓	✓	09:03	09:03	09:05	09:05	0.0	0.0	0.0	0.0
RQ_9ab3	08:27	09:08	✓	✓	09:16	09:16	09:17	09:17	3.2	3.2	0.0	0.0
RQ_6cbc	08:30	09:09	✓	✓	09:35	09:35	09:42	09:42	7.5	7.5	0.5	0.5
RQ_dd11	08:30	09:10	✓	✓	09:10	09:10	09:16	09:16	0.0	0.0	1.4	1.4
RQ_3522	08:31	09:07	✓	✓	09:11	09:11	09:14	09:14	0.0	0.0	0.0	0.0
RQ_e92f	08:36	09:11	✓	✓	09:26	09:26	09:28	09:28	5.9	5.9	0.0	0.0
RQ_e91f	08:39	09:16	✓	✓	09:34	09:34	09:40	09:40	6.4	6.4	0.5	0.5
RQ_789f	08:41	09:10	✓	✓	09:20	09:20	09:22	09:22	4.9	4.9	0.0	0.0
RQ_e85d	08:42	09:17	✓	✓	09:18	09:18	09:19	09:19	0.0	0.0	0.0	0.0
RQ_1f8e	08:46	09:31	×	✓	—	09:41	—	09:44	—	5.3	—	0.0
RQ_49e0	08:49	09:09	✓	✓	09:44	09:44	09:46	09:46	7.8	7.8	0.0	0.0
RQ_96af	08:50	09:05	✓	✓	09:07	09:07	09:09	09:09	0.0	0.0	0.0	0.0
RQ_3933	08:54	09:27	×	✓	—	09:47	—	09:52	—	5.7	—	0.0
RQ_0abf	08:54	09:28	✓	✓	09:32	09:32	09:34	09:34	0.0	0.0	0.0	0.0
RQ_c17a	08:55	09:27	✓	✓	09:47	09:53	09:51	09:57	5.5	11.4	0.0	0.2
RQ_40fb	08:57	09:21	×	✓	—	09:47	—	09:56	—	5.5	—	5.1
RQ_1c92	09:00	09:26	✓	✓	09:28	09:28	09:32	09:32	0.0	0.0	0.0	0.0
RQ_da0e	09:08	09:25	✓	✓	09:43	09:48	09:45	09:51	6.2	6.9	0.0	0.0
RQ_5a33	09:15	09:46	✓	✓	09:49	09:54	09:51	09:56	0.0	2.9	0.0	0.0
RQ_544e	09:22	09:38	✓	✓	09:42	09:42	09:43	09:43	0.0	0.0	0.0	0.0
RQ_0699	09:23	10:00	✓	✓	10:00	10:00	10:02	10:02	0.0	0.0	0.0	0.0
RQ_ef6e	09:32	10:10	✓	✓	10:10	10:10	10:15	10:15	0.0	0.0	0.3	0.3
RQ_ac00	09:36	10:07	✓	✓	10:07	10:07	10:11	10:11	0.0	0.0	0.0	0.0
RQ_211b	09:41	10:14	✓	✓	10:14	10:14	10:19	10:19	0.0	0.0	0.0	0.0
RQ_e2c6	09:51	10:16	✓	✓	10:20	10:20	10:33	10:33	0.0	0.0	7.0	7.0
RQ_e358	09:52	10:23	✓	✓	10:23	10:23	10:27	10:27	0.0	0.0	0.0	0.0
RQ_b547	09:52	10:12	✓	✓	10:12	10:12	10:15	10:15	0.0	0.0	0.2	0.2
RQ_81ad	09:55	10:20	✓	✓	10:23	10:23	10:27	10:27	0.0	0.0	0.0	0.0
RQ_a466	09:58	10:28	✓	✓	10:32	10:32	10:38	10:38	0.0	0.0	0.0	0.0
RQ_6a04	10:03	10:36	✓	✓	10:39	10:39	10:45	10:45	0.0	0.0	3.5	3.5
RQ_aa92	10:04	10:26	✓	✓	10:29	10:29	10:31	10:31	0.0	0.0	0.0	0.0
RQ_9b46	10:07	10:36	✓	✓	10:44	10:44	10:50	10:47	3.3	3.3	3.6	0.0
RQ_bd66	10:11	10:40	✓	✓	10:40	10:40	10:42	10:42	0.0	0.0	0.0	0.0
RQ_397b	10:17	10:33	✓	✓	10:41	10:41	10:42	10:42	2.3	2.3	0.0	0.0
RQ_dacf	10:23	11:02	✓	✓	11:02	11:06	11:04	11:07	0.0	0.0	0.0	0.0
RQ_b6fe	10:23	10:53	✓	✓	10:56	10:53	11:04	11:01	0.0	0.0	0.0	0.0
RQ_5aa1	10:24	11:06	✓	✓	11:07	11:14	11:18	11:21	0.0	2.9	6.9	2.8

Table 8 Request-level rolling-horizon results for a simulated workday (WO0.LMPS8.1: open objective, staggered start; cont. 1/2). Pickup and delivery times represent the realized service-start times. Delay denotes pickup tardiness, while excess ride time is measured relative to the direct trip duration. Accepted and rejected requests are denoted by ✓ and ×, respectively.

Key	Submission time	Earliest pickup	Accepted?		Pickup time		Delivery time		Delay (min)		Excess ride (min)	
			MILP3	ALNS	MILP3	ALNS	MILP3	ALNS	MILP3	ALNS	MILP3	ALNS
RQ_5e21	10:26	10:47	✓	✓	10:47	10:51	10:53	10:55	0.0	0.0	2.6	0.0
RQ_3c5e	10:34	11:12	✓	✓	11:12	11:15	11:16	11:19	0.0	0.0	0.0	0.3
RQ_8a13	10:35	11:08	✓	✓	11:14	11:14	11:17	11:17	1.6	1.6	0.0	0.0
RQ_6642	10:38	11:06	✓	✓	11:06	11:09	11:11	11:15	0.0	0.0	0.0	0.6
RQ_0bba	11:02	11:43	✓	✓	11:43	11:43	11:48	11:48	0.0	0.0	0.0	0.0
RQ_1ff3	11:35	12:03	✓	✓	12:03	12:03	12:06	12:06	0.0	0.0	0.0	0.0
RQ_65ba	11:39	12:18	✓	✓	12:18	12:18	12:20	12:20	0.0	0.0	0.0	0.0
RQ_830d	11:43	12:03	✓	✓	12:03	12:03	12:06	12:06	0.0	0.0	0.0	0.0
RQ_b1df	11:58	12:24	✓	✓	12:24	12:24	12:27	12:27	0.0	0.0	0.0	0.0
RQ_db21	12:11	12:51	✓	✓	12:51	12:55	12:52	12:56	0.0	0.0	0.0	0.0
RQ_a49b	12:13	12:58	✓	✓	13:06	13:00	13:09	13:02	3.6	0.0	0.2	0.2
RQ_e121	12:17	12:46	✓	✓	12:50	12:54	12:55	12:59	0.0	2.9	1.3	1.2
RQ_519d	12:20	12:51	✓	✓	12:58	13:02	13:05	13:08	2.3	6.2	3.3	2.3
RQ_c28d	12:26	12:49	✓	✓	12:58	12:49	13:02	12:53	4.1	0.0	0.0	0.0
RQ_5db2	12:26	12:45	✓	✓	12:45	12:45	12:48	12:48	0.0	0.0	0.0	0.0
RQ_0813	12:32	13:15	✓	✓	13:23	13:19	13:24	13:19	3.3	0.0	0.0	0.0
RQ_f8c4	12:33	13:06	✓	✓	13:12	13:10	13:19	13:17	1.1	0.0	1.0	1.0
RQ_2e51	12:34	13:12	✓	✓	13:28	13:23	13:31	13:27	8.8	5.7	2.1	2.1
RQ_da88	12:35	12:51	✓	✓	13:10	13:04	13:15	13:13	14.7	8.8	1.1	4.9
RQ_d86d	12:36	13:02	✓	✓	13:08	13:03	13:12	13:06	1.7	0.0	0.6	0.4
RQ_8fbd	12:38	13:19	✓	✓	13:44	13:28	13:49	13:34	7.7	3.5	0.0	0.0
RQ_0a82	12:38	13:12	✓	✓	13:35	13:19	13:37	13:22	6.1	1.7	0.0	0.0
RQ_0c14	12:46	13:25	✓	✓	13:30	13:25	13:32	13:27	0.0	0.0	0.2	0.2
RQ_c55c	13:03	13:22	✓	✓	13:35	13:30	13:45	13:36	5.8	2.7	3.6	0.0
RQ_7371	13:07	13:41	✓	✓	13:41	13:41	13:46	13:43	0.0	0.0	2.7	0.0
RQ_31f3	13:11	13:39	✓	✓	13:43	13:39	13:53	13:43	0.0	0.0	5.8	0.0
RQ_3748	13:13	13:43	✓	✓	13:49	13:46	13:55	13:50	0.2	0.0	2.1	0.0
RQ_1601	13:14	13:54	✓	✓	13:55	13:54	13:59	13:58	0.0	0.0	0.0	0.0
RQ_9b66	13:31	13:59	✓	✓	14:01	13:59	14:07	14:05	0.0	0.0	0.0	0.0
RQ_0e68	13:40	14:16	✓	✓	14:16	14:16	14:18	14:18	0.0	0.0	0.0	0.0
RQ_256c	13:46	14:15	✓	✓	14:15	14:15	14:18	14:18	0.0	0.0	0.0	0.0
RQ_0a32	14:00	14:45	✓	✓	14:45	14:45	14:50	14:50	0.0	0.0	0.0	0.0
RQ_6cb9	14:20	14:45	✓	✓	14:45	14:45	14:48	14:48	0.0	0.0	0.0	0.0
RQ_ed5b	14:32	14:55	✓	✓	14:55	14:55	15:01	15:01	0.0	0.0	0.0	0.0
RQ_4318	14:33	14:56	✓	✓	14:56	14:56	14:58	14:58	0.0	0.0	0.0	0.0
RQ_2c9b	14:51	15:33	✓	✓	15:36	15:33	15:39	15:35	0.0	0.0	0.0	0.0
RQ_926e	14:57	15:17	✓	✓	15:17	15:17	15:20	15:20	0.0	0.0	0.0	0.0
RQ_9f9d	15:02	15:28	✓	✓	15:28	15:28	15:32	15:32	0.0	0.0	0.0	0.0
RQ_faf9	15:34	15:54	✓	✓	15:54	15:54	15:56	15:56	0.0	0.0	0.0	0.0

Table 9 Request-level rolling-horizon results for a simulated workday (WO0.LMPS8.1: open objective, staggered start; cont. 2/2). Pickup and delivery times represent the realized service-start times. Delay denotes pickup tardiness, while excess ride time is measured relative to the direct trip duration. Accepted and rejected requests are denoted by ✓ and ×, respectively.

The performed experiments establish three complementary findings:

- Our proposed feasibility-driven insertion procedure provides substantial computational savings as route length and constraint restrictiveness increase.
- Exact formulations exhibit superior performance for small and medium-sized instances, yet performance rapidly deteriorates as problem size increases. Conversely, ALNS consistently delivers competitive solutions across all investigated problem sizes.
- The rolling-horizon study demonstrates that solver choice, routing objective, and deployment strategy jointly influence full-day performance. Our experiments indicate that ALNS and staggered starts improve operational robustness, while the open objective reduces resource consumption per served request.

7 Conclusion and Future Work

In this work, we presented a routing framework tailored for dynamic, on-demand mobility services. We formulated the routing problem as an e-ADARP variant, for which we developed three MILP models capturing key operational features, such as heterogeneous agents, request selectivity, soft TWs, and extended SoC representation. Furthermore, we introduced a feasibility-driven insertion heuristic coupled with an ALNS algorithm to enhance scalability and real-time performance. Our approach supports dynamically emerging requests through a rolling-horizon framework, which decomposes the dynamic problem into static snapshots that are subsequently solved by the routing strategies.

We conducted a series of numerical experiments to evaluate the developed methods, including a feasible-insertion ablation study, a static four-way solver benchmark, and dynamic full-workday simulations for MILP3 and ALNS. The results demonstrate substantial computational gains from the proposed insertion procedure and emphasize the combined impacts of solver choice, routing objective, and fleet-deployment strategy on full-workday service and resource efficiency. In addition, we proposed a PoC for the full-stack software solution, encompassing its architecture, software components, and communication middleware. Future research will focus on the implementation and deployment of MODEMS, followed by real-life experiments on campus grounds. We anticipate that longer request-submission horizons, including multi-day advance submissions or early-day demand forecasts, may provide stronger partial plans throughout the workday, improve anticipation of demand surges, and support the dynamic staggering of agent start times.

Declarations

Research Funding

This work is part of project MORE (Munich Mobility Research Campus) and funded by dttec.bw – Digitalization and Technology Research Center of the Bundeswehr. dttec.bw is funded by the European Union – NextGenerationEU.

The funding was provided within the broader scope of the MORE project; no specific resources were dedicated to the work presented in this manuscript, and no

additional research grants or commercial collaborations contributed to its preparation or influenced its content.

References

- [1] Pobudzei, M., Huemer, A.K., Hoffmann, S.: Uncovering micromobility patterns in non-urban landscapes: Insights from the more sharing system. In: 2024 Forum for Innovative Sustainable Transportation Systems (FISTS), pp. 1–8. IEEE, Riverside, CA, USA (2024). <https://doi.org/10.1109/fists60717.2024.10485609>
- [2] Pobudzei, M., Hoffmann, S.: User segmentation based on usage frequency: A case study of a multimodal shared micromobility in a non-urban campus environment. In: 2024 IEEE Intelligent Vehicles Symposium (IV), pp. 992–999. IEEE, Jeju Island, Korea, Republic of (2024). <https://doi.org/10.1109/iv55156.2024.10588699>
- [3] Braekers, K., Ramaekers, K., Van Nieuwenhuyse, I.: The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering* **99**, 300–313 (2016) <https://doi.org/10.1016/j.cie.2015.12.007>
- [4] Ojeda Rios, B.H., Xavier, E.C., Miyazawa, F.K., Amorim, P., Curcio, E., Santos, M.J.a.: Recent dynamic vehicle routing problems: A survey. *Computers & Industrial Engineering* **160**, 107604 (2021) <https://doi.org/10.1016/j.cie.2021.107604>
- [5] Cordeau, J.-F., Laporte, G.: The dial-a-ride problem: models and algorithms. *Annals of Operations Research* **153**(1), 29–46 (2007) <https://doi.org/10.1007/s10479-007-0170-8>
- [6] Qin, H., Su, X., Ren, T., Luo, Z.: A review on the electric vehicle routing problems: Variants and algorithms. *Frontiers of Engineering Management* **8**(3), 370–389 (2021) <https://doi.org/10.1007/s42524-021-0157-1>
- [7] Schneider, M., Stenger, A., Goeke, D.: The electric vehicle-routing problem with time windows and recharging stations. *Transportation Science* **48**(4), 500–520 (2014) <https://doi.org/10.1287/trsc.2013.0490>
- [8] Molenbruch, Y., Braekers, K., Caris, A.: Typology and literature review for dial-a-ride problems. *Annals of Operations Research* **259**(1-2), 295–325 (2017) <https://doi.org/10.1007/s10479-017-2525-0>
- [9] Su, Y., Parragh, S.N., Dupin, N., Puchinger, J.: The Bi-objective Electric Autonomous Dial-a-Ride Problem (2025). <https://doi.org/10.48550/arXiv.2512.16605>
- [10] Su, Y., Dupin, N., Parragh, S.N., Puchinger, J.: A branch-and-price algorithm for the electric autonomous dial-a-ride problem. *Transportation Research Part B: Methodological* **186**, 103011 (2024) <https://doi.org/10.1016/j.trb.2024.103011>

- [11] Stallhofer, V., Parragh, S.N.: Event-based models for the electric autonomous dial-a-ride problem. *Transportation Research Part C: Emerging Technologies* **171**, 104896 (2025) <https://doi.org/10.1016/j.trc.2024.104896>
- [12] Wang, S., Sun, W., Huang, M.: An adaptive large neighborhood search for the multi-depot dynamic vehicle routing problem with time windows. *Computers & Industrial Engineering* **191**, 110122 (2024) <https://doi.org/10.1016/j.cie.2024.110122>
- [13] Gaul, D., Klamroth, K., Stiglmayr, M.: Solving the dynamic dial-a-ride problem using a rolling-horizon event-based graph. In: Müller-Hannemann, M., Perea, F. (eds.) *21st Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2021)*. Open Access Series in Informatics (OASICS), vol. 96, pp. 8–1816. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2021). <https://doi.org/10.4230/OASICS.ATMOS.2021.8>
- [14] Zhang, J., Woensel, T.V.: Dynamic vehicle routing with random requests: A literature review. *International Journal of Production Economics* **256** (2023) <https://doi.org/10.1016/j.ijpe.2022.108751>
- [15] Tóth, M., Hajba, T., Horváth, A.: Milp models of a patient transportation problem. *Central European Journal of Operations Research* (2024) <https://doi.org/10.1007/s10100-023-00902-z>
- [16] Rist, Y., Forbes, M.A.: A new formulation for the dial-a-ride problem. *Transportation Science* **55**(5), 1113–1135 (2021) <https://doi.org/10.1287/trsc.2021.1044>
- [17] Tellez, O., Vercraene, S., Lehuédé, F., Péton, O., Monteiro, T.: The fleet size and mix dial-a-ride problem with reconfigurable vehicle capacity. *Transportation Research Part C: Emerging Technologies* **91**, 99–123 (2018) <https://doi.org/10.1016/j.trc.2018.03.020>
- [18] Ropke, S., Pisinger, D.: An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science* **40**(4), 455–472 (2006) <https://doi.org/10.1287/trsc.1050.0135>
- [19] Gschwind, T., Drexel, M.: Adaptive large neighborhood search with a constant-time feasibility test for the dial-a-ride problem. *Transportation Science* **53**(2), 480–491 (2019) <https://doi.org/10.1287/trsc.2018.0837>
- [20] Keskin, M., Laporte, G., Çatay, B.: Electric vehicle routing problem with time-dependent waiting times at recharging stations. *Computers & Operations Research* **107**, 77–94 (2019) <https://doi.org/10.1016/j.cor.2019.02.014>
- [21] Gasset, D., Paillalef, F., Payacán, S., Gatica, G., Herrera-Vidal, G., Linfati, R., Coronado-Hernández, J.R.: Route optimization for open vehicle routing problem (ovrp): A mathematical and solution approach. *Applied Sciences* **14**(16) (2024)

<https://doi.org/10.3390/app14166931>

- [22] Voigt, S.: A review and ranking of operators in adaptive large neighborhood search for vehicle routing problems. *European Journal of Operational Research* **322**(2), 357–375 (2025) <https://doi.org/10.1016/j.ejor.2024.05.033>
- [23] Ho, S.C., Szeto, W.Y., Kuo, Y.-H., Leung, J.M.Y., Petering, M., Tou, T.W.H.: A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research Part B: Methodological* **111**, 395–421 (2018) <https://doi.org/10.1016/j.trb.2018.02.001>
- [24] Bemporad, A., Morari, M.: Control of systems integrating logic, dynamics, and constraints. *Automatica* **35**(3), 407–427 (1999) [https://doi.org/10.1016/S0005-1098\(98\)00178-2](https://doi.org/10.1016/S0005-1098(98)00178-2)
- [25] Williams, H.P.: *Model Building in Mathematical Programming*, 5th edn. Wiley-Blackwell, Hoboken, NJ (2013)
- [26] Gaul, D., Klamroth, K., Stiglmayr, M.: Event-based milp models for ridepooling applications. *European Journal of Operational Research* **301**(3), 1048–1063 (2022) <https://doi.org/10.1016/j.ejor.2021.11.053>
- [27] Xu, W., Zhang, C., Cheng, M., Huang, Y.: Electric vehicle routing problem with simultaneous pickup and delivery: Mathematical modeling and adaptive large neighborhood search heuristic method. *Energies* **15**(23), 9222 (2022) <https://doi.org/10.3390/en15239222>
- [28] Sayarshad, H.R., Mahmoodian, V., Gao, H.O.: Non-myopic dynamic routing of electric taxis with battery swapping stations. *Sustainable Cities and Society* **57**, 102113 (2020) <https://doi.org/10.1016/j.scs.2020.102113>
- [29] Sumitkumar, R., Al-Sumaiti, A.S.: Shared autonomous electric vehicle: Towards social economy of energy and mobility from power-transportation nexus perspective. *Renewable and Sustainable Energy Reviews* **197**, 114381 (2024) <https://doi.org/10.1016/j.rser.2024.114381>
- [30] Bongiovanni, C.: The electric autonomous dial-a-ride problem. PhD thesis, École Polytechnique Fédérale de Lausanne (EPFL) (2020). <https://doi.org/10.5075/epfl-thesis-7431>
- [31] Pfeiffer, C., Schulz, A.: An alns algorithm for the static dial-a-ride problem with ride and waiting time minimization. *OR Spectrum* **44**(1), 87–119 (2021) <https://doi.org/10.1007/s00291-021-00656-7>
- [32] Zhang, X., Dai, X., Lou, P., Hu, J.: A multi-strategy alns for the vrp with flexible time windows and delivery locations. *Applied Sciences* **15**(9), 4995 (2025) <https://doi.org/10.3390/app15094995>

- [33] Karich, P.: Flexible Routenplanung mit GraphHopper. FOSSGIS e.V. (2016). <https://doi.org/10.5446/19732> . <https://av.tib.eu/media/19732>
- [34] Marie, P.: Modern Web Cartography: Open Source Map Solutions with Open-StreetMap, Overpass, Nominatim, Leaflet and Folium. Apress, Berkeley, CA (2026). <https://doi.org/10.1007/979-8-8688-2335-0>
- [35] Qu, Y., Bard, J.F.: The heterogeneous pickup and delivery problem with configurable vehicle capacity. *Transportation Research Part C: Emerging Technologies* **32**, 1–20 (2013) <https://doi.org/10.1016/j.trc.2013.03.007>
- [36] Macenski, S., Foote, T., Gerkey, B., Lalancette, C., Woodall, W.: Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics* **7**(66), 6074 (2022) <https://doi.org/10.1126/scirobotics.abm6074>
- [37] Bynum, M.L., Hackebeil, G.A., Hart, W.E., Laird, C.D., Nicholson, B.L., Siirola, J.D., Watson, J.-P., Woodruff, D.L.: *Pyomo – Optimization Modeling in Python* vol. 67, 3rd edn. Springer, Switzerland (2021). <https://doi.org/10.1007/978-3-030-68928-5>
- [38] Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2024). <https://www.gurobi.com>
- [39] Wouda, N.A., Lan, L.: ALNS: a Python implementation of the adaptive large neighbourhood search metaheuristic. *Journal of Open Source Software* **8**(81), 5028 (2023) <https://doi.org/10.21105/joss.05028>
- [40] Barker, M., Chue Hong, N.P., Katz, D.S., Lamprecht, A.-L., Martinez-Ortiz, C., Psomopoulos, F., Harrow, J., Castro, L.J., Gruenpeter, M., Martinez, P.A., Honeyman, T.: Introducing the fair principles for research software. *Scientific Data* **9**(1) (2022) <https://doi.org/10.1038/s41597-022-01710-x>