The Dynamic Park-and-Loop Routing Problem

Nicolás Cabrera[a], Jean-François Cordeau[a], Jorge E. Mendoza[a]

[a]*HEC Montréal, 3000 chemin de la Côte-Sainte-Catherine, Montréal, H3T 2A7 Canada*

Abstract

This paper introduces the dynamic park-and-loop routing problem, an important practical extension of the park-and-loop routing problem in which some of the customer demand is revealed dynamically during the planning horizon. Each time a new customer request is received, the current plan must be re-evaluated and, in many cases, revised to accommodate the new request. The objective is to maximize the number of requests served. To address this problem, we propose a set of anticipatory scheduling policies that build on the multi-space sampling heuristic. We evaluate these policies on a new set of instances with up to 100 requests and assess their performance using a dual bound based on the perfect information relaxation. Our results show the large benefits of anticipation in both offline and real-time routing decisions. We also provide a comprehensive analysis of the problem parameters leading to valuable insights.

Keywords: Park-and-loop, dynamic vehicle routing, stochastic requests, multi-space sampling heuristic

1. Introduction

Rapid urbanization, increasing traffic congestion, and the scarcity of curb space have significantly complicated the execution of short-range service operations in dense city environments (Holguín-Veras et al., 2020). In many metropolitan areas, a growing share of operational tasks—such as the removal of improperly parked shared vehicles, infrastructure inspections,

light maintenance, and on-site sample collection—must be performed within limited-access neighborhoods where parking opportunities are scarce and vehicle circulation is constrained. These operational contexts require service teams to combine vehicle movements with short walking or micromobility trips in order to efficiently reach dispersed customer locations.

In such settings, it is often inefficient to drive directly to every service location. Instead, operators commonly adopt a *park-and-loop* strategy: a vehicle is parked at a strategic location, and a set of nearby requests is served sequentially on foot or using a lightweight mobility device before returning to the vehicle. This structure reduces vehicle circulation in dense areas, mitigates parking search time, and limits exposure to traffic congestion (Reed et al., 2024). Park-and-loop operations are increasingly relevant in applications such as shared bicycle and scooter rebalancing, where crews must retrieve, reposition, or remove vehicles from sidewalks and pedestrian zones, as well as in urban utility and inspection services operating in restricted-access districts (Schuijbroek et al., 2017; Shui & Szeto, 2020).

While park-and-loop routing has been studied in static settings (Coindreau et al., 2019; Cabrera et al., 2022; Le Colleter et al., 2023; Cabrera et al., 2023; Jodiawan et al., 2025), many real-world operations unfold in highly dynamic environments. A subset of service requests may be known in advance (e.g., planned inspections or scheduled rebalancing tasks), whereas others arise dynamically throughout the day (e.g., newly reported improperly parked vehicles, urgent inspection needs, or demand-triggered repositioning tasks). As new requests are revealed, operators must decide whether to accept them and how to adapt current routes accordingly. Crucially, earlier parking decisions affect future flexibility: choosing where to park determines the set of requests that can be efficiently incorporated later. This interdependence between parking, walking subtours, and dynamic request arrivals creates a challenging sequential decision problem.

We refer to this problem as the *Dynamic Park-and-Loop Routing Problem* (DPLRP). The DPLRP combines three fundamental elements: (i) a two-level routing structure composed of vehicle tours and walking subtours, (ii) a mix of scheduled and on-demand requests, and (iii) sequential acceptance and routing decisions made under uncertainty. Solving the DPLRP requires balancing immediate operational efficiency with future adaptability. On the one hand, initial routes must efficiently cover scheduled tasks. On the other hand, sufficient flexibility must be preserved to accommodate future requests without incurring excessive detours or additional parking operations.

The literature on dynamic vehicle routing has extensively investigated online insertion strategies, anticipatory policies based on sampled scenarios, and waiting strategies that preserve future flexibility (Bent & Van Hentenryck, 2004, 2007; Ulmer et al., 2018, 2019; Zhang & Van Woensel, 2023). However, most existing approaches focus on standard vehicle routing structures in which customers are served directly by vehicle. In contrast, the DPLRP introduces additional structural decisions: parking locations, the composition and sequencing of walking subtours, and the potential reassignment of requests between workers as the system evolves. Updating a solution may involve extending or shortening an existing walking subtour, inserting a new subtour at a different parking location, reordering requests within subtours, or reoptimizing entire routes. These features substantially increase the complexity of dynamic adjustments.

To address these challenges, we develop a family of non-anticipatory and anticipatory scheduling policies based on the multi-space sampling heuristic (MSH), a route-first, assemble-second metaheuristic originally proposed for stochastic vehicle routing problems (Mendoza & Villegas, 2013). At each decision epoch, our approach generates a diverse set of high-quality routes that account for the current system state, including vehicle positions and partially completed subtours. A set partitioning model is then used to as-

semble a feasible solution from this route pool. By repeatedly solving this structured subproblem as new requests arrive, the proposed framework enables rich inter-route and intra-route adjustments while maintaining computational tractability in an online setting.

Beyond methodological development, this paper provides a quantitative assessment of the operational value of flexibility in dynamic park-and-loop systems. In particular, we introduce a perfect information bound that defines the maximum number of requests that could be served if all future arrivals were known in advance. This bound allows us to evaluate the benefits of anticipatory decision making. Furthermore, we generate a new set of instances derived from the street network of Vienna, Austria, extending the testbed introduced by Zhang et al. (2023). All instances, results, and a solution checker are made publicly available to foster future research on dynamic routing in parking-constrained urban environments.

The remainder of this paper is organized as follows. Section 2 reviews the related literature. Section 3 formally defines the DPLRP and presents a route-based sequential decision process. Section 4 describes the proposed scheduling policies. Section 5 introduces the perfect information bound and its solution approach. Section 6 reports the computational experiments. Section 7 concludes the paper and discusses directions for future research.

2. Literature review

In this section, we first review the literature on problems related to the DPLRP. This is followed by an overview of solution methods.

2.1. Park-and-loop and parking-aware routing

Routing problems that combine vehicle movements with walking subtours arise naturally in dense urban environments where curb space is limited and direct vehicle access to every service location is impractical. In such settings,

operators often park the vehicle at a strategic location and serve several nearby customers on foot before returning to the vehicle. This operational structure is commonly referred to as a park-and-loop route. These problems share a common structural feature: the vehicle does not directly serve all customers along its route, but instead temporarily disengages from service execution to allow an alternative service mode, such as walking or cycling. This decoupling between vehicle positioning and service execution is a central abstraction underlying park-and-loop and related routing problems.

One of the earliest problems exhibiting this structure is the truck-and-trailer routing problem (TTRP) introduced by Semet (1995) and later studied by Chao (2002). In the TTRP, trucks pulling trailers serve a set of customers, some of which cannot be accessed with the trailer attached. To address this constraint, the trailer may be detached at designated decoupling points, after which the truck serves nearby customers on its own before returning to the trailer. Although the operational context differs from pedestrian service settings, the key idea of temporarily separating the vehicle (or a part of it) from service execution is shared with park-and-loop routing. Because all customers are known in advance, the TTRP is typically addressed using static optimization methods, including heuristics (Derigs et al., 2013; Villegas et al., 2013) and exact algorithms (Belenguer et al., 2016; Parragh & Cordeau, 2017; Rothenbächer et al., 2018).

A more direct example of this decoupling structure appears in park-and-loop variants of the traveling salesman problem. The two-echelon last-mile delivery problem (2E-LMDP) studied by Martinez-Sykora et al. (2020) can be interpreted as a simplified park-and-loop routing problem in which a single vehicle performs one park-and-walk operation to serve all customers. The authors propose an exact branch-and-cut algorithm capable of solving instances with up to 30 customers. In this setting, the walking component is restricted to a single subtour originating from a given parking location.

This constraint has been relaxed in subsequent work. Reed et al. (2024) introduced the capacitated delivery problem with parking (CDPP), which extends the 2E-LMDP by allowing the worker to perform an arbitrary number of walking subtours starting from the same parking position. This evolution reflects a broader trend in park-and-loop modeling, where increasingly flexible settings are considered.

Another related example arises in vehicle routing problems involving auxiliary mobile service units, such as drones. In the two-echelon vehicle routing problem with time windows and mobile satellites (2EVRP-TM) introduced by Li et al. (2020), each vehicle carries a limited number of drones that can be dispatched either from the depot or from customer locations after the vehicle is parked. Once all drones return, the vehicle route continues. While the physical service mechanism differs from walking subtours, this problem similarly separates vehicle movement decisions from service execution decisions. To solve this problem, the authors propose an adaptive large neighborhood search (ALNS) heuristic capable of handling instances with up to 100 customers. More recently, Zhou et al. (2023) presented a branch-and-price algorithm for a variant in which the number of drones per vehicle is itself a decision variable.

This line of work has culminated in the park-and-loop routing problem (PLRP), a term initially coined by Le Colleter et al. (2023), which explicitly models routes composed of vehicle segments with walking subtours. In this framework, workers may either drive to a location or park the vehicle and serve nearby customers on foot. Several solution approaches have been proposed for the static PLRP. Coindreau et al. (2019) proposed a variable neighborhood search algorithm combined with a fast insertion mechanism. Cabrera et al. (2022) developed a metaheuristic based on the multi-space sampling heuristic. Le Colleter et al. (2023) introduced a hybrid small and large neighborhood search heuristic. In a similar vein, Cabrera et al. (2023)

proposed an exact branch-price-and-cut algorithm capable of solving most benchmark instances with up to 50 customers. In all these studies, however, customer requests are assumed to be known in advance and routing decisions are computed before the start of operations. More recently, Jodiawan et al. (2025) proposed a variant of the PLRP that incorporates explicit parking considerations and additional flexibility in route design, solved using a hybrid large neighborhood search algorithm.

Following this line of research, Scherr et al. (2026) introduced the dynamic same-day delivery problem with walking and parking (SDDPWP), which considers a single-vehicle same-day delivery setting where customer requests arrive dynamically over time. In this problem, the driver may park the vehicle and serve multiple customers on foot before returning to continue the route. The authors focus on dispatch timing decisions and propose threshold-based policies that determine when delivery tours should be initiated from the depot. Each routing plan is computed by solving a CDPP to optimality. Their computational experiments highlight the trade-off between customer consolidation, parking time, and outsourcing decisions in dense urban environments. While this work also considers dynamic routing decisions with parking and walking operations, it focuses on a single-vehicle dispatch setting with threshold-based policies. In contrast, the DPLRP addresses a multi-worker routing setting.

Despite these recent advances, the dynamic coordination of multiple workers performing park-and-loop operations remains largely unexplored. In particular, limited attention has been devoted to anticipatory scheduling policies capable of dynamically reoptimizing parking and walking decisions across multiple routes as new requests arrive over time. The DPLRP addresses this challenge by integrating parking, walking, and routing decisions where customer requests are progressively revealed.

2.2. Dynamic routing and sequential decision models

The DPLRP also relates to the broader literature on dynamic vehicle routing problems (DVRPs), in which routing decisions must be continuously updated as new customer requests appear during the planning horizon. The DVRP has been widely studied in the operations research literature (Pillac et al., 2012).

Various algorithmic strategies have been proposed to address the sequential nature of the problem. Scenario-based approaches anticipate future requests by sampling possible realizations of demand (Bent & Van Hentenryck, 2004). Waiting and relocation strategies aim to preserve future flexibility by allowing vehicles to remain idle or relocate to areas with higher expected demand (Bent & Van Hentenryck, 2007). Other approaches rely on real-time optimization and metaheuristics to dynamically update routing plans as new information becomes available (Ferrucci et al., 2013; Sarasola et al., 2016). More recently, anticipatory decision-making methods based on simulation have been proposed to approximate optimal policies under uncertainty (Ulmer et al., 2018).

Despite this extensive literature, most dynamic routing models assume that customers are served directly by vehicle. In contrast, the DPLRP introduces additional structural decisions related to parking locations and walking subtours. These features significantly increase the complexity of dynamically updating routing plans as new requests arrive.

2.3. Solution methods

To solve dynamic vehicle routing problems, researchers have explored many alternatives. Arguably the most common method is known as *reoptimization*. Reoptimization updates the current routing plan by solving a *static* version of the problem with the most recent available information. Typically, this is done by using metaheuristics, as solutions are to be obtained in relatively short computational times. Depending on how often the

static problem is solved, reoptimization methods can be classified into periodic (PR) and continuous (CR). As the name suggests, PR methods solve the static problem at fixed intervals of time (e.g., every 5 minutes) or after observing a pre-defined number of events (e.g., after five customer requests are received). For example, Pillac et al. (2018) developed a fast metaheuristic that is applied every time a new customer request is revealed. Alternatively, CR methods run continuously during the planning horizon. To do so, these methods often rely on maintaining a pool of solutions that can be adapted to the newly disclosed information without significant effort. For instance, Gendreau et al. (1999) presented a parallel tabu search to continuously improve the set of feasible solutions. This method is executed in multiple threads that are only interrupted when new information is available.

Another line of research models dynamic vehicle routing problems as Markov decision processes (MDPs). An MDP is well-suited for modeling as it explicitly incorporates uncertainty and stepwise planning (Ulmer, 2017). However, these models face the well-known *curse of dimensionality* (Powell, 2007), as real-world problems involve vast state, action, and outcome spaces, rendering exact solution methods computationally infeasible. As a result, researchers have focused on the design of approximate dynamic programming (ADP) methods. The simplest ADPs are myopic policies, which, as the name suggests, ignore future customer requests. Instead, they focus solely on maximizing immediate rewards, often resulting in poor performance. To overcome this, more complex methods that explicitly sample and simulate dynamic requests have been developed. These methods can be classified into three categories: rollout algorithms, policy function approximation (PFA), and value function approximation (VFA).

Rollout algorithms, introduced by Bertsekas et al. (1997), leverage an existing policy, known as the *base policy*, to improve decision-making in dynamic programming problems. The intuition behind these methods is to *roll*

out the base policy by simulating multiple future steps starting from the current state. These simulations generate potential outcomes for different actions, enabling an evaluation of their quality based on the expected cumulative reward over the simulated horizon. For instance, Secomandi (2001) developed a rollout algorithm for the vehicle routing problem with stochastic demands, using the cyclic heuristic as the base policy. Their results demonstrated performance improvements of up to 4% on instances with up to 40 customers. However, a key limitation of rollout algorithms is that all simulations are performed online, which can result in significant delays in real-time decision-making. In this paper, we use a rollout algorithm as one of our benchmark algorithms.

In a search for efficiency, researchers have developed policy function approximation methods (Ulmer & Thomas, 2018; Ulmer & Streng, 2019). These methods use decision rules designed to emulate effective decision-making, typically relying on thresholds determined through offline simulations. For instance, Ulmer & Thomas (2018) proposed a PFA for solving the dynamic vehicle routing problem with heterogeneous fleets of drones and vehicles. Their approach divides the service area into two zones based on vehicle travel time from the depot. Customers within a specified threshold distance are served by vehicles, while those beyond it are served by drones. Similarly, Ulmer & Streng (2019) designed a PFA for same-day delivery systems involving pickup stations and autonomous vehicles. This method determines when and where to dispatch vehicles to pickup stations and which goods to load, using a threshold to balance fast delivery with consolidation efficiency. More recently, Scherr et al. (2026) proposed threshold-based dispatch policies for a dynamic same-day delivery problem with walking and parking. Their approach relies on parameterized decision rules that determine when a vehicle should initiate a delivery tour based on the number of accepted customers or the average number of customers served per park-

ing stop. Since these policies directly approximate the dispatching strategy without explicitly estimating downstream rewards through simulation, they can be viewed as policy function approximations. Since the thresholds are pre-defined, PFA methods are highly efficient. However, they are only suitable for problems in which a good policy can be readily observed (Powell, 2007).

Value function approximation is another common approach (Ulmer et al., 2019; Ulmer & Thomas, 2020; Ulmer et al., 2020; Soeffker et al., 2024). This method estimates the value of being in a particular state or the expected value of taking a specific action. Due to the typically large number of states, they are often aggregated into a set of features. Ulmer et al. (2018) proposed a non-parametric value function based on the available time to approximate the cost of routing decisions. Their solution incorporates a dynamic lookup table that adaptively partitions the state space during the learning process. Soeffker et al. (2024) introduced parametric VFAs that model vehicle workloads at varying levels of detail, enabling effective customer request assignments. Similarly, Ulmer & Thomas (2020) developed a meso-parametric VFA (M-VFA) that integrates non-parametric and parametric VFAs, leveraging their strengths while mitigating their limitations. In a related effort, Ulmer et al. (2019) combined a VFA with a rollout algorithm to address the dynamic vehicle routing problem with stochastic requests. This hybrid approach demonstrated the benefits of blending offline and online methods to generate higher-quality policies.

The primary drawback of VFAs lies in their high computational complexity, as these methods often rely on lookup tables that are susceptible to the curse of dimensionality. To address this issue, Zhang et al. (2023) proposed a set of knapsack-based linear models that compute the expected reward of a decision without requiring offline simulations. In these models, each vehicle is represented as a knapsack, where the capacity corresponds to the remain-

ing service time available in the vehicle's shift. By combining these models with an intra-route routing policy, they developed efficient online scheduling policies. Under these policies, once a request is assigned to a vehicle, it cannot be reassigned. Their experiments on large-scale, real-time instances demonstrated that these policies outperformed traditional ADP approaches in both solution quality and computational efficiency. Building on this concept, we adapt the knapsack-based linear models and propose scheduling policies that leverage the multi-space sampling heuristic to enable efficient inter-route operations. Our experiments indicate that our scheduling policies enhance solution quality with minimal impact on computational efficiency.

Table 1 presents a classification of the literature related to this paper. Column 1 provides a reference for each article. Columns 2 and 3 indicate whether or not the article considers routing and parking decisions, respectively. Column 4 shows if the article considers uncertainty in the problem parameters, such as the travel times (T) or the demand (D). Column 5 shows if the method proposed in the article anticipates the impact of a decision in the future costs or rewards. Column 6 reports the method used to build the initial solution. Column 7 shows the method that updates the initial solution.

3. Problem definition

In this section, we formally define the DPLRP. We first present the problem description. We then present a route-based sequential decision process inspired by the route-based Markov decision processes introduced by Ulmer et al. (2020).

3.1. Problem description

The DPLRP can be formally defined on a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, where $\mathcal{N}$ is the node set and $\mathcal{A}$ is the arc set. The set $\mathcal{N}$ comprises the depot represented by node 0 and the customer request set $\mathcal{C}$. Each customer

Table 1: Literature classification.

Author	Decisions		Uncertainty	Anticipation	Offline policy	Online policy
	Routing	Parking				
Semet (1995)	✓	✓			TS	n/a
Chao (2002)	✓	✓			TS	n/a
Derigs et al. (2013)	✓	✓			LNS	n/a
Villegas et al. (2013)	✓	✓			GRASP	n/a
Belenguer et al. (2016)	✓	✓			B&C	n/a
Parragh & Cordeau (2017)	✓	✓			BAP	n/a
Rothenbächer et al. (2018)	✓	✓			BPC	n/a
Coindreau et al. (2019)	✓	✓			VNS	n/a
Li et al. (2020)	✓	✓			ALNS	n/a
Martinez-Sykora et al. (2020)	✓	✓			B&C	n/a
Cabrera et al. (2022)	✓	✓			MSH	n/a
Cabrera et al. (2023)	✓	✓			BPC	n/a
Le Colleter et al. (2023)	✓	✓			SLNS	n/a
Zhou et al. (2023)	✓	✓			BAP	n/a
Reed et al. (2024)	✓	✓			CI	n/a
Jodiawan et al. (2025)	✓	✓			HLNS	n/a
Gendreau et al. (1999)	✓		D		TS	TS
Bent & Van Hentenryck (2004)	✓		D	✓	MSA	MSA
Mitrović-Minić & Laporte (2004)	✓		D		n/a	TS
Bent & Van Hentenryck (2007)	✓		D	✓	LNS	LNS
Ferrucci et al. (2013)	✓		D		TS	TS
Ulmer et al. (2018)	✓		D	✓	CI	PFA+rollout
Pillac et al. (2018)	✓		D		ALNS	ALNS
Ulmer et al. (2019)	✓		D	✓	CI	VFA+rollout
Ulmer & Thomas (2020)	✓		D	✓	n/a	VFA
Ulmer et al. (2020)	✓		D	✓	n/a	VFA
Zhang et al. (2023)	✓		D	✓	PB	PB
Scherr et al. (2026)	✓	✓	D	✓	CI	PFA
This work	✓	✓	D	✓	PB-MSH	PB-MSH

request $i \in \mathcal{C}$ has a fixed duration ν_i. To serve customer requests, a set $\mathcal{W}$ of homogeneous workers is available. Arcs in $\mathcal{A}$ represent the connections between two locations. Each arc (i, j) has four main attributes: the walking distance α_{ij}, the driving distance μ_{ij}, the walking time γ_{ij}, and the driving time δ_{ij}. We assume that every node $i \in \mathcal{N}$ is a feasible parking location, i.e., a worker may park a vehicle at any node, incurring a parking time τ. The maximum walking distance between two nodes is Υ_{max}, the maximum total walking distance per worker over the planning horizon is Υ^{day}, and the maximum duration of a walking subtour is Γ.

Set $\mathcal{C}$ can be divided into two disjoint subsets. The first subset contains the scheduled requests $\mathcal{C}^s$, which are known at the beginning of the planning horizon. These requests must be served before the end of the planning horizon. The second set contains the on-demand requests $\mathcal{C}^d$, which are not known in advance. We assume these requests appear randomly during the

working hours $[e, f]$. For each new customer request, a decision is made to either accept or reject it. If the request is accepted, it must be served within the planning horizon. On the contrary, if rejected, it is assumed to be outsourced or postponed to a later date.

We also define a route plan θ as a set of park-and-loop routes $\{r_1, \ldots, r_{|\mathcal{W}|}\}$ for each worker. A park-and-loop route $r = \{0, i_1, i_2, \ldots, i_{|r|-2}, 0\}$ is an ordered sequence of parking locations visited by the vehicle, starting and ending at the depot. For each parking location $i_h \in r$, the worker parks the vehicle to serve request i_h and may subsequently perform a walking subtour $u_h = (i_h, j_1, \ldots, j_{m_h}, i_h)$, where $m_h \geq 0$ and $j_1, \ldots, j_{m_h}$ is the ordered sequence of additional requests served on foot before returning to the vehicle at i_h. When $m_h = 0$, the worker serves only the parking-location request i_h, and no additional requests are served on foot; the walking subtour then reduces to $u_h = (i_h, i_h)$. The total duration and the total walking distance of route r are defined by T_r and υ_r.

The objective of the DPLRP is to design a route plan that maximizes the number of on-demand requests served such that: the total duration of each route does not exceed the workday duration; the total walking distance for each worker does not exceed the walking distance limit; and the duration of each walking subtour does not exceed the time limit. An illustrative example of an instance of the DPLRP is included in Appendix A.

3.2. Route-based sequential decision process

In this section, we present a route-based sequential decision process that captures the different forms of uncertainty in the DPLRP.

3.2.1. Decision epochs

Decision epochs $k \in \{0, \ldots, K\}$ refer to specific points in time when decisions are made. In the DPLRP, a decision is made every time that an on-demand request i_k appears. In addition, a decision is made at the start

of the planning horizon to design the initial routing plan. The number of decision points per day K is a random variable.

3.2.2. States

A state represents the current status of the system at a specific decision epoch k. We define the state of the system at decision epoch k as $\mathcal{S}_k$. In our problem, the state $\mathcal{S}_k$ consists of the following eight components:

- the current time denoted by $t_k \in \mathbb{R}_{\geq 0}$,
- the current routing plan denoted by θ_k,
- the current destination of each worker w denoted by $\hat{\imath}_{wk} \in \mathcal{N}$,
- the arrival time of each worker w at their current destination denoted by $\hat{\jmath}_{wk} \in [e, f]$,
- the remaining allowable time in the current walking subtour for each worker w, denoted by $\hat{u}_{wk} \in [0, \Gamma]$, i.e., the time left before reaching the maximum subtour duration,
- the distance that can still be walked by each worker w denoted by $l_{wk} \in [0, \Upsilon^{day}]$,
- the current parking spot of the vehicle of each worker w denoted by $p_{wk} \in \mathcal{N}$,
- and the set of accepted but not yet served requests denoted by $\mathcal{C}_k$.

State $\mathcal{S}_k$ can be summarized as $\mathcal{S}_k = (t_k, \theta_k, \hat{\imath}_k, \hat{\jmath}_k, \hat{u}_k, l_k, p_k, \mathcal{C}_k)$ with $\hat{\imath}_k$, $\hat{\jmath}_k$, $\hat{u}_k$, l_k, and p_k being $|\mathcal{W}|$-dimensional vectors. If a worker is serving a request, the value of $\hat{\imath}_{wk}$ is set to the worker's current location. In addition, if a worker is not performing a subtour at time t_k, the value of $\hat{u}_{wk}$ is set to 0. Also, if the worker is driving between two locations, p_{wk} is set to the current destination. In the initial state $\mathcal{S}_0 = (0, \theta_0, \hat{\imath}_0, \hat{\jmath}_0, \hat{u}_0, l_0, p_0, \mathcal{C}_0)$,

all workers are idle at the depot and $\mathcal{C}_0 = \mathcal{C}^s$. To guarantee feasibility, we assume that there exists a routing plan that can serve all the requests in $\mathcal{C}^s$. At the final decision epoch K, the process is at a terminal state $\mathcal{S}_K = (t_K, \theta_K, \hat{\imath}_K, \hat{\jmath}_K, \hat{u}_K, l_K, p_K, \varnothing)$, where all the workers are back at the depot and all accepted requests are served.

3.2.3. Actions

Actions are the decisions made at each decision epoch k that influence the state of the system. At state $\mathcal{S}_0$ the set of feasible decisions $\mathcal{F}$ is composed of all the feasible routing plans θ that include the set of scheduled requests $\mathcal{C}^s$. We represent the initial decision by $\theta_0 \in \mathcal{F}$. This decision is made offline prior to the start of the planning horizon. For the remaining decision epochs $k \neq 0$, decisions are made online. Given $\mathcal{S}_k$, an acceptance decision is made about whether to accept or reject the new on-demand request i_k. In case the new on-demand request is accepted, it is also necessary to make a routing decision according to which the routing plan θ_k is updated. There are only two possible acceptance decisions, but there are many possible routing decisions.

We denote by $\mathcal{X}_k$ the set of feasible decisions at state $\mathcal{S}_k$. Each decision $x_k \in \mathcal{X}_k$ specifies both the acceptance and the routing decisions. If the new on-demand request is rejected, the routing decision defaults to preserving the current routing plan. Formally, a decision is a tuple $x_k = (\mathbb{I}_k^x, \theta_k^x)$, where $\mathbb{I}_k^x$ is an indicator variable that takes the value of 1 if the on-demand request i_k is accepted and θ_k^x is the updated routing plan that must include all the requests that are yet to be served: $\mathcal{C}_k^x = \mathcal{C}_k \cup \{i_k\}$. A decision is feasible when the following conditions hold:

- the park-and-loop routes $r \in \theta_k^x$ are feasible,
- the workers driving or walking between two locations pursue their current destinations,

- and the accepted request is assigned to exactly one worker.

3.2.4. Transitions

When a decision x_k is made at decision epoch k, it leads to the state transition from the pre-decision state $\mathcal{S}_k$ to post-decision state $\mathcal{S}_k^x = (t_k^x, \theta_k^x, \hat{\imath}_k^x, \hat{\jmath}_k^x, \hat{u}_k^x, l_k^x, p_k^x, \mathcal{C}_k^x)$. In $\mathcal{S}_k^x$ we update the workers' destinations and arrival times of the previously idle workers. These values are inherited from the updated routing plan θ_k^x. Similarly, we update the available time for performing a walking subtour and the remaining distance to be covered on foot. We also update the current parking spots for each worker. Finally, the set of accepted but not yet served requests is updated by removing the requests served between decision epochs $k-1$ and k, and by adding request i_k in case an accept decision was made. The transition from the post-decision state $\mathcal{S}_k^x$ to the next pre-decision state $\mathcal{S}_{k+1} = (t_{k+1}, \theta_{k+1}, \hat{\imath}_{k+1}, \hat{\jmath}_{k+1}, \hat{u}_{k+1}, l_{k+1}, p_{k+1}, \mathcal{C}_{k+1})$ occurs at the time of the $(k+1)$-th decision epoch t_{k+1}. The transition updates the state as follows:

- the routing plan θ_{k+1} is obtained by removing the requests served before t_{k+1} from θ_k^x,
- the set of accepted but not yet served requests is modified as $\mathcal{C}_{k+1} = \mathcal{C}_k^x \setminus \tilde{\mathcal{C}}$, where $\tilde{\mathcal{C}}$ corresponds to the request served before time t_{k+1},
- and a new on-demand request i_{k+1} appears.

3.2.5. Rewards

The reward quantifies the immediate benefit (or cost) of an action. We define the reward of a decision x_k as a marginal reward corresponding to the difference between the value of the current route plan and the updated one. Let $CR(\theta)$ denote the cumulative reward associated with route plan θ. Also, let $\tilde{\mathbb{I}}_k$ be an indicator variable equal to 1 if the on-demand request i_k is accepted, and 0 otherwise.

The initial route plan only contains the requests known a priori. Therefore, at the initial epoch ($k = 0$), the cumulative reward is defined as:

$$CR(\theta_0) = |\mathcal{C}^s|. \tag{1}$$

For any decision epoch $k \geq 1$, the cumulative reward of route plan θ_k is defined as:

$$CR(\theta_k) = |\mathcal{C}^s| + \sum_{i=1}^{k-1} \tilde{\mathbb{I}}_i. \tag{2}$$

Here, $|\mathcal{C}^s|$ represents the constant base reward associated with the requests known a priori, which must be served regardless of the dynamic policy.

After decision x_k is taken at decision epoch k, the cumulative reward of the updated routing plan θ_k^x is:

$$CR(\theta_k^x) = CR(\theta_k) + \mathbb{I}_k^x. \tag{3}$$

The reward $R(\cdot)$ associated with decision x_k is then defined as the marginal increase in cumulative reward:

$$R(\mathcal{S}_k, x_k) = CR(\theta_k^x) - CR(\theta_k). \tag{4}$$

3.2.6. Objective

Let Π denote the set of Markovian deterministic scheduling policies, where a policy $\pi \in \Pi$ is a sequence of decision rules $(X_0^\pi, X_1^\pi, \ldots, X_K^\pi)$ to map a state $\mathcal{S}_k$ to decision $x_k = X_k^\pi(\mathcal{S}_k)$. These decision rules may be identical for every epoch $k \in \{1, \ldots, K\}$. We seek an optimal scheduling policy

$\pi^* \in \Pi$ that maximizes the expected total reward beginning from the initial state $\mathcal{S}_0$:

$$V(\mathcal{S}_0) = \max_{\pi \in \Pi} \mathbb{E}\left[\sum_{k=0}^{K} R(\mathcal{S}_k, X_k^{\pi}(\mathcal{S}_k))|\mathcal{S}_0\right]. \tag{5}$$

In the context of the DPLRP, a scheduling policy comprises an offline policy that defines the initial routing plan θ_0, an acceptance policy that decides whether to accept or reject a request, and a routing policy that updates the routing plan after the acceptance decisions. In the following section, we present different alternatives for each of these policies.

4. Scheduling policies

In this section, we describe the building blocks of our scheduling framework. To address the different decision stages of the DPLRP, we define a complete scheduling policy π as a triplet $(\vartheta, \rho, \varrho)$, representing the offline initialization, the real-time routing strategy, and the acceptance logic, respectively. Figure 1 illustrates the algorithmic building blocks we develop and shows how they can be combined to assemble different scheduling policies. In Appendix C we provide a summary of our set of scheduling policies.

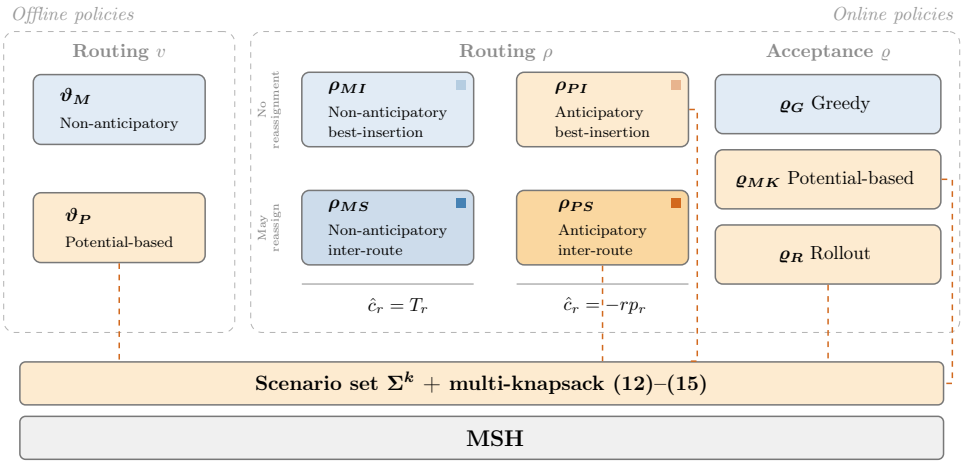


Figure 1: Overview of scheduling policies.

Both the offline and routing policies proposed in this work rely on a common optimization framework based on the multi-space sampling heuristic (MSH). The MSH acts as a generic route generation and assembly procedure that can be adapted to different decision contexts through the definition of route generation functions and assembly objectives. For this reason, we first describe the MSH framework and the scenario-based approximation used to evaluate future service potential.

Then, we detail the three components of our framework in the following order:

1. Offline policies (ϑ): which define the initial routing plan (Section 4.3).
2. Routing policies (ρ): which determine how workers are mobilized to serve requests (Section 4.4).
3. Acceptance policies (ϱ): which determine whether newly arriving requests are served (Section 4.5).

4.1. Multi-space sampling heuristic

The multi-space sampling heuristic (MSH) is the core optimization framework used by several offline and online routing policies proposed in this work. At each decision epoch, the MSH generates a set of candidate park-and-loop routes using problem-specific route generation functions and combines them into a routing plan through an assembly procedure. Different offline and routing policies emerge from the definition of the route generation functions included in Ξ and from the objective optimized by the assembler.

The MSH comprises two phases, namely, a route generation phase and an assembly phase. In the route generation phase, the algorithm iteratively builds a pool (i.e., a set) of high-quality routes by using a set of route generation functions. A route generation function can, for example, take the form of a local search procedure that improves an existing route. In the assembly phase, the algorithm solves a set partitioning model that builds a final solu-

tion. This step is carried out by an assembler. An example illustrating the two phases of the MSH is included in Appendix B.

Algorithm 1 presents the main logic of the MSH. Line 1 initializes the pool of park-and-loop routes $\overline{\Omega}_w$ for each worker $w \in \mathcal{W}$. Then, we enter the outer loop (lines 2-9) where the algorithm populates each pool of routes by iterating over a set of route generation functions Ξ. This outer loop is executed in parallel by ζ independent threads. Line 3 initializes the iteration number. Afterward, the algorithm uses the inner loop (lines 5-7) for a given number of iterations per thread (Q) or a maximum execution time (T_{rge}^{max}). Line 5 computes a subset of routes r_w^q for each worker using the selected route generation function. Line 6 adds the routes found to the corresponding pool of routes. Line 10 calls the assembler to recombine routes in the pool into the final solution. This solution is stored in $\tilde{\Omega}_w$. Finally, Line 11 returns the final solution $\tilde{\Omega}_w$.

Algorithm 1 `MSH` function

Require: $\mathcal{G}$, graph; Ξ, route generation functions; Q, iteration limit; T_{rge}^{max}, time limit route generation phase; T_{asm}^{max}, time limit assembly phase.
Ensure: final solution $\tilde{\Omega}_w$

1: $\overline{\Omega}_w \leftarrow \emptyset$
2: **for** $\xi \in \Xi$ **do**
3: $\quad q \leftarrow 1$
4: $\quad$ **while** $q < \frac{Q}{|\Xi|} \wedge route_generation_time < T_{rge}^{max}$ **do**
5: $\quad\quad r_w^q \leftarrow \xi(\mathcal{G})$
6: $\quad\quad \overline{\Omega}_w \leftarrow \overline{\Omega}_w \cup r_w^q$
7: $\quad\quad q \leftarrow q + 1$
8: $\quad$ **end while**
9: **end for**
10: $\tilde{\Omega}_w \leftarrow \texttt{Assembly}(\mathcal{G}, \overline{\Omega}_w, T_{asm}^{max})$
11: **return** $\tilde{\Omega}_w$

Depending on the definition of the set of route generation functions Ξ and the assembler, different scheduling policies emerge. In what follows, we provide a general definition of the assembler. The assembler can be generally defined as follows. Let Ω_w^k define a subset of feasible park-and-loop routes

for each worker given their status at decision epoch k. Let $\hat{c}_r$ be the cost of route r. Also, let a_{ir} be a binary parameter that takes the value 1 if route r serves request i, and 0 otherwise. Let y_r be a binary variable that takes the value 1 if route r is selected, and 0 otherwise. Then, the assembler at any decision epoch $k > 0$ is defined as:

$$\min \sum_{w \in \mathcal{W}} \sum_{r \in \Omega_w^k} \hat{c}_r y_r \tag{6}$$

subject to

$$\sum_{w \in \mathcal{W}} \sum_{r \in \Omega_w^k} a_{ir} y_r = 1 \qquad \forall i \in \mathcal{C}_k \tag{7}$$

$$\sum_{w \in \mathcal{W}} \sum_{r \in \Omega_w^k} a_{ir} y_r \leq 1 \qquad i = i_k \tag{8}$$

$$\sum_{r \in \Omega_w^k} y_r = 1 \qquad \forall w \in \mathcal{W} \tag{9}$$

$$y_r \in \{0, 1\} \qquad \forall w \in \mathcal{W}, r \in \Omega_w^k. \tag{10}$$

The objective function (6) minimizes the total cost of the routing plan. Constraints (7) ensure that all the scheduled and previously accepted requests are included in the routing plan. Constraint (8) states that if accepted, the new on-demand request can only be assigned to one route. Constraints (9) guarantee that one route is assigned to each worker. Formulation (6)-(10) can be easily adapted to handle additional constraints. Moreover, the definition of the cost $\hat{c}_r$ for each route can be redefined to accomplish different goals. For example, one could consider the total travel time of a route as its cost, which would likely result in selecting shorter routes.

The assembler formulation defines how candidate routes generated by the MSH are combined into a routing plan. In the following subsection, we introduce a scenario-based approximation of the expected reward-to-go that is used to evaluate the future service potential of routing plans within

anticipatory routing and acceptance policies.

4.2. Value function approximation and scenario-based evaluation

In dynamic routing settings, decisions must be made sequentially under uncertainty regarding future customer requests. At each decision epoch k, the system must evaluate the impact of current decisions on future performance. To formalize this, we define the expected reward-to-go at state $\mathcal{S}_k$ under a scheduling policy π as:

$$\hat{V}_\pi(\mathcal{S}_k) = \mathbb{E}\left[\sum_{i=k+1}^{K} R(\mathcal{S}_i, X_i^\pi(\mathcal{S}_i)) \mid \mathcal{S}_k\right]. \tag{11}$$

This quantity represents the expected cumulative reward from decision epoch k until the end of the planning horizon. However, computing this expectation exactly is intractable due to the stochastic nature of request arrivals and the size of the state space.

To efficiently approximate this quantity, we leverage the multi-knapsack approximation introduced by Zhang et al. (2023). The intuition behind this approximation is to estimate how many future customer requests can be feasibly inserted into the current routing plan θ_k, given the remaining time available on each route. Let Σ^k denote a set of sampled scenarios, where each scenario $\sigma \in \Sigma^k$ represents a realization of future customer requests from decision epoch k to the end of the planning horizon. Also, let $\mathcal{C}_\sigma$ denote the set of customer requests that arrive in scenario $\sigma \in \Sigma^k$. Let η_{ir} be the minimum increase in the duration of route $r \in \theta_k$ obtained by inserting request $i \in \mathcal{C}_\sigma$. We define binary decision variables z_{ir} that take the value 1 if request i is assigned to route r, and 0 otherwise. The multi-knapsack approximation for scenario σ is given by:

$$Z^\sigma(\mathcal{S}_{k+1}) = \max \sum_{i \in \mathcal{C}_\sigma} \sum_{r \in \theta_k} z_{ir} \tag{12}$$

subject to

$$\sum_{i \in \mathcal{C}_\sigma} \eta_{ir} z_{ir} \leq f - T_r \qquad \forall r \in \theta_k \tag{13}$$

$$\sum_{r \in \theta_k} z_{ir} \leq 1 \qquad \forall i \in \mathcal{C}_\sigma \tag{14}$$

$$z_{ir} \in \{0, 1\} \qquad \forall i \in \mathcal{C}_\sigma, r \in \theta_k. \tag{15}$$

The objective function maximizes the number of requests that can be inserted into the routing plan. Constraints (13) ensure that the total additional time required to serve inserted requests does not exceed the remaining time available on each route. Constraints (14) guarantee that each request is assigned to at most one route. Following Zhang et al. (2023), we relax the integrality constraints on variables z_{ir} in order to speed up computation.

The expected reward-to-go can then be approximated by averaging over all scenarios:

$$\hat{V}(\mathcal{S}_k) \approx \frac{1}{|\Sigma^k|} \sum_{\sigma \in \Sigma^k} Z^\sigma(\mathcal{S}_{k+1}). \tag{16}$$

This approximation provides a computationally efficient proxy for evaluating the future service potential of a routing plan. In the remainder of this section, it is used (i) to guide routing decisions through potential-based routing policies, and (ii) to evaluate acceptance decisions via scenario-based approximations.

4.3. Offline policies

A key step to solving the DPLRP is to build a high-quality initial routing plan at the initial state $\mathcal{S}_0$. In this section, we describe two offline policies that are implemented using the MSH. These policies differ in how route generation functions are defined and in the objective used within the assembler.

4.3.1. Non-anticipatory offline routing policy

The first offline policy only considers the set of scheduled requests $\mathcal{C}^s$. Let R be the set of all feasible park-and-loop routes that serve only the scheduled requests. Also, let a_{ir} be a parameter that takes the value 1 if route r serves request i, and 0 otherwise. Let c_r be the total distance covered using the vehicle in route r. Finally, let y_r be a binary variable that takes the value 1 if route r is selected as part of the initial routing plan and 0 otherwise. Then, an initial routing plan can be computed using the following set partitioning formulation:

$$\min \sum_{r \in R} c_r y_r \tag{17}$$

subject to

$$\sum_{r \in R} a_{ir} y_r = 1 \qquad \forall i \in \mathcal{C}^s \tag{18}$$

$$\sum_{r \in R} y_r \leq |\mathcal{W}| \tag{19}$$

$$y_r \in \{0, 1\} \qquad \forall r \in R. \tag{20}$$

The objective function (17) minimizes the total distance covered by the vehicles. Constraints (18) guarantee that all scheduled requests are served. Constraints (19) impose a limit on the number of routes that are selected. Constraints (20) state the decision variables are binary. Generating the complete set of routes R may not be possible if the number of scheduled requests $|\mathcal{C}^s|$ is large. Thus, exactly solving the mathematical formulation (17)-(20) is not always feasible in practice. Instead, one may rely on metaheuristics or column generation-based methods to generate a high-quality set of routes. In the context of the DPLRP, we define the route generation function $\xi^g(\mathcal{C}^s)$ that builds a subset of routes $\overline{R}$ using as input the scheduled requests. This route generation function starts by building a traveling

salesman problem (TSP) tour that includes all scheduled requests. Then, the TSP tour is partitioned into routes using the split algorithm proposed by Cabrera et al. (2022). To generate TSP tours, we use four classic TSP construction heuristics, namely, randomized nearest neighbor (RNN), randomized nearest insertion (RNI), randomized farthest insertion (RFI), and randomized best insertion (RBI).

The non-anticipatory offline routing policy ϑ_M uses the set of route generation functions defined as $\Xi = \bigcup_{g \in G} \xi^g(\mathcal{C}^s)$, where $G = \{RNN, RBI, RFI, RNI\}$. For the assembler that builds the initial routing plan, this policy uses formulation (17)-(20).

4.3.2. Potential-based offline routing policy

The previous policy may result in the assignment of very long routes to some workers, and very short ones to others. Some workers may even remain idle at the depot for some time, as the result of not being assigned to any of the routes. A shortcoming of this behavior is that in the long run, a higher number of on-demand requests will be rejected. Thus, in a similar fashion to the potential-based online routing policies, we use a set of scenarios Σ^0 to compute the potential of each route rp_r. To do so, we solve the multi-knapsack model (12)-(15) for each scenario in set Σ^0. Using the notation presented in the previous section, we can compute an initial routing plan with the following formulation:

$$\max \sum_{r \in R} rp_r y_r \tag{21}$$

subject to (18)-(20).

The objective function (21) maximizes the total potential of the initial routing plan. The potential-based offline routing policy ϑ_P uses the set of route generation functions defined as $\Xi = \bigcup_{g \in G} \xi^g(\mathcal{C}^s)$, where $G = \{RNN, RBI, RFI, RNI\}$. Again, the assembler that builds the initial rout-

ing plan solves formulation (18)-(21).

4.4. Routing policies

A routing policy determines how workers are mobilized to serve requests. In this work, routing decisions are made at each decision epoch using the MSH. At each decision epoch, the MSH generates candidate routes through a set of route generation functions and selects a routing plan via an assembly procedure that depends on the chosen policy. Our first two routing policies are non-anticipatory. That is, they ignore that new customer requests may appear before the end of the planning horizon. Instead, they aim to minimize the total travel time. In contrast, our third routing policy is anticipatory. That is, it aims to maximize the expected number of served customer requests. In what follows, we describe these policies in detail.

4.4.1. Non-anticipatory best-insertion routing policy

The first routing policy relies on the best insertion route generation function $\xi^{BI}(r, i_k) \in \Xi$. This function constructs a new route r' by inserting request i_k into an existing route r through the creation of a new parking location associated with a dedicated walking subtour. More precisely, r' is obtained by selecting the position in r that minimizes the insertion cost while preserving the order of existing parking locations. Let $\mathcal{A}_r^d$ be the ordered set of driving arcs of route r. The insertion cost $c^d_{i_k jl}$ corresponds to inserting a new parking location serving request i_k between two consecutive parking locations $(j, l) \in \mathcal{A}_r^d$. It is defined as: $c^d_{i_k jl} = \delta_{j i_k} + \delta_{i_k l} - \delta_{jl}$. If the insertion is not feasible, i.e., if the resulting route violates the time budget $T_r + c^d_{i_k jl} + \nu_{i_k} + \tau > f$, the insertion cost is set to infinity. Note that this route generation function assumes that each newly inserted request is served via a dedicated walking subtour originating from a newly created parking location. It does not consider inserting requests into existing walking subtours.

To overcome this issue, we define the cost of inserting a new request

i_k into an existing walking subtour u_h. The walking arcs (in order) within each subtour are denoted by $\mathcal{A}_{u_k}$. Let $\mathcal{T}_{u_h}$ denote the total duration of the walking subtour u_h. If $\mathcal{A}_{u_h}$ is empty (i.e., $m_h = 0$), the insertion corresponds to starting a new walking subtour from the associated parking location i_h, with cost $c^w_{i_k i_h} = \gamma_{i_h i_k} + \gamma_{i_k i_h} + \nu_{i_k}$. Otherwise, if $m_h > 0$, the insertion cost of request i_k into u_h is: $c^e_{i_k u_h} = \min_{(j,l)\in\mathcal{A}_{u_h}} \{\gamma_{ji_k} + \gamma_{i_k l} - \gamma_{jl}\} + \nu_{i_k}$. Let (j^*, l^*) denote the insertion arc that minimizes the insertion cost $c^e_{i_k u_h}$. Then, the corresponding increase in walking distance is given by $\alpha^{\Delta}_{i_k u_h} = \alpha_{j^* i_k} + \alpha_{i_k l^*} - \alpha_{j^* l^*}$.

We then define the route generation function $\xi^{ES}(r, i_k) \in \Xi$, which assigns request i_k to the feasible walking subtour that minimizes the insertion cost. An insertion is deemed infeasible if $c^e_{i_k u_h} + \mathcal{T}_{u_h} > \Gamma \quad \text{or} \quad c^e_{i_k u_h} + T_r > f$ or $\alpha^{\Delta}_{i_k u_h} + \upsilon_r > \Upsilon^{day}$. In this case, the cost is set to infinity. Arcs (i, j) with $\alpha_{ij} > \Upsilon_{max}$ are excluded during preprocessing. Note that the route generation functions $\xi^{BI}(r, i_k)$ and $\xi^{ES}(r, i_k)$ do not modify the order of already accepted requests, nor do they consider requests outside those already served by the current route. If no feasible insertion exists, the corresponding route generation function returns $\emptyset$.

Using route generation functions $\xi^{BI}(r, i_k)$ and $\xi^{ES}(r, i_k)$ we define the non-anticipatory best-insertion routing policy ρ_{MI} in which the set of route generation functions is defined as $\Xi = \bigcup_{r\in\theta} \xi^{BI}(r, i_k) \cup \bigcup_{r\in\theta} \xi^{ES}(r, i_k)$. Moreover, the cost $\hat{c}_r$ of each route in the assembler is set to the total duration of the route T_r.

4.4.2. Non-anticipatory inter-route routing policy

Our second routing policy allows for inter-route operations. That is, it allows for the reassignment of accepted requests between workers. It also allows for more complex operations such as extending or shortening existing walking subtours, and the insertion of new ones if convenient. To do so, we define the route generation function $\xi^g(w, \mathcal{C}_k, i_k)$. This route generation

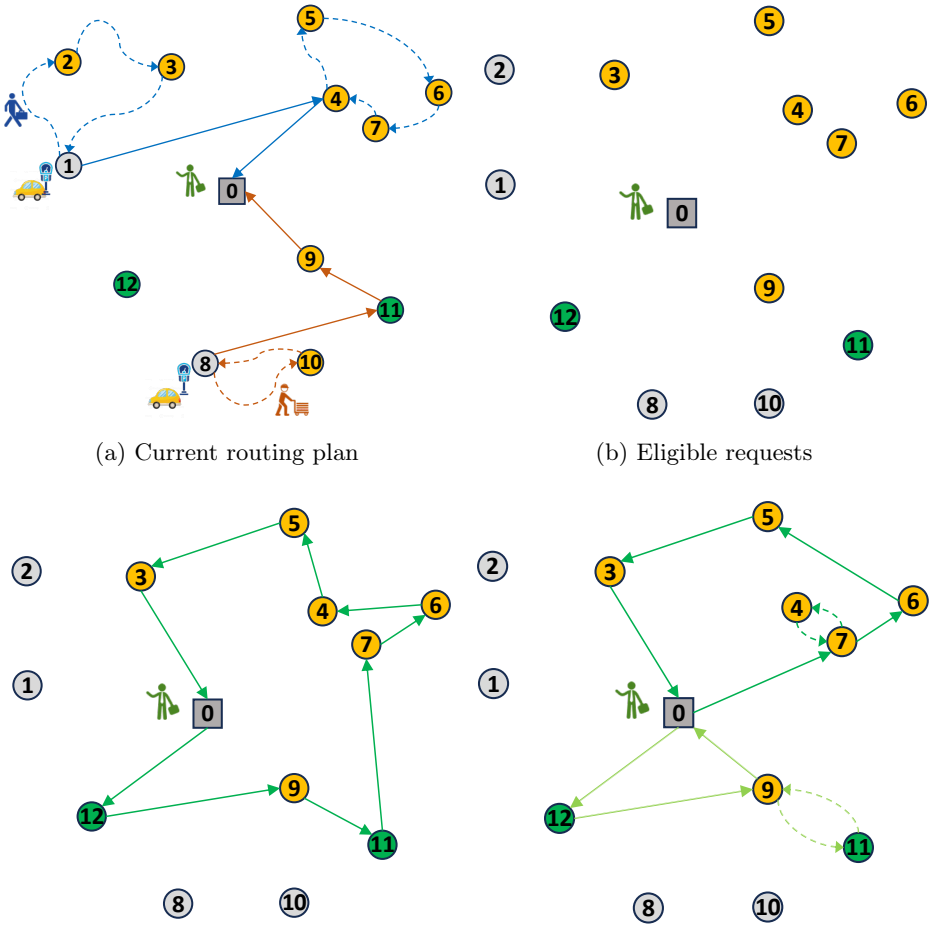
function creates a new subset of routes Ω_w for worker w using as input the set $\mathcal{C}_k$ of accepted but not yet served requests, and the new on-demand request i_k. The route generation function $\xi^g(w, \mathcal{C}_k, i_k)$ relies on two components. First, we use a randomized traveling salesman problem heuristic that creates paths from the worker's current location to the depot, including both the accepted but not yet served requests and the new on-demand request. Second, we use a tailored split algorithm introduced by Cabrera et al. (2022) that partitions the path into feasible routes.

Figure 2 illustrates how route generation function $\xi^g(w, \mathcal{C}_k, i_k)$ operates. Figure 2a presents the current state of the system, including the location of an idle worker highlighted in green. Figure 2b displays the eligible requests, i.e., the accepted but not served requests and the new on-demand Request 12. Figure 2c shows a feasible path that begins at the worker's current location (i.e., the depot) and visits all accepted but not yet served requests, including customers already assigned to a route but not yet visited (e.g., Customers 1, 2, 8, and 10), as well as the new on-demand Request 12. Finally, Figure 2d shows two park-and-loop routes obtained after using the split algorithm, one of which includes the new on-demand request.

The inter-route routing policy ρ_{MS} uses the set of route generation functions defined as $\Xi = \bigcup_{w \in \mathcal{W}, g \in G} \xi^g(w, \mathcal{C}_k, i_k)$, where $G = \{RNN, RBI, RFI, RNI\}$. The cost of each route in the assembler $\hat{c}_r$ is set to the total duration of the route T_r.

4.4.3. Potential-based routing policy

The routing policies described in Sections 4.4.1 and 4.4.2 ignore the arrival of new requests and instead focus on immediate benefits. As a result, they may yield suboptimal long-term decisions, where routes that seem efficient at first become inefficient over time. We propose a potential-based routing policy that builds on the multi-knapsack approximation model (12)-(15) and aims to remedy these issues. This policy takes into account the



(a) Current routing plan

(b) Eligible requests

(c) Path with eligible requests

(d) Feasible routes

Figure 2: Inter-route route generation function example.

expected number of requests that can potentially be inserted into a given park-and-loop route. In particular, let us again consider the set of scenarios Σ^k. Let us also define the potential rp_r of each route r as the expected number of on-demand requests that can be feasibly inserted. The potential rp_r of a route r can be computed by setting $\theta = \{r\}$ and solving formulation (12)-(15). This amounts to setting the routing plan to be equal to a single route r. Note that in this case constraints (14) can be removed.

Using this definition, we define two potential-based routing policies that improve on policies ρ_{MI} and ρ_{MS}. The key difference with the non-anticipatory

policies is that the cost of each route in the assembler $\hat{c}_r$ is set to the potential of the route rp_r multiplied by -1. These two policies are denoted as ρ_{PI} and ρ_{PS}. Using these policies, one can expect the design of routing plans that would tend to better accommodate new on-demand requests. However, it must be noted that computing the potential of each route in the subset of routes generated by the route generation functions can be computationally expensive if the number of routes is too large.

4.5. Acceptance policies

Acceptance policies determine whether a newly arriving on-demand request is inserted into the current routing plan or rejected.

At each decision epoch k, upon the arrival of a new request, the system must decide whether accepting the request leads to a better expected outcome than rejecting it. Let $\mathcal{S}_k^+$ and $\mathcal{S}_k^-$ denote the post-decision states corresponding to acceptance and rejection, respectively. The acceptance decision is defined as:

$$\mathbb{I}_k^x = \mathbb{I}\big(1 + \hat{V}_\pi(\mathcal{S}_k^+) \geq \hat{V}_\pi(\mathcal{S}_k^-)\big). \tag{22}$$

In practice, the expected reward-to-go $\hat{V}_\pi(\cdot)$ is approximated using the scenario-based multi-knapsack formulation introduced in Section 4.2. In the following, we describe three acceptance policies: a greedy baseline that ignores future arrivals, and two anticipatory policies that use the approximation in different ways.

4.5.1. Greedy acceptance policy

The greedy acceptance policy ϱ_G accepts a new request if it can be feasibly inserted into the current routing plan using any admissible routing policy (see Section 4.4). This policy ignores future request arrivals and therefore serves as a baseline.

4.5.2. Potential-based acceptance policy

The potential-based acceptance policy ϱ_{MK} evaluates the acceptance rule (22) using the multi-knapsack approximation introduced in Section 4.2. In particular, the expected reward-to-go of each post-decision state is approximated by the expected number of future requests that can be feasibly inserted into the routing plan over the set of scenarios Σ^k.

A request is accepted if the approximated reward associated with $\mathcal{S}_k^+$ is greater than or equal to that of $\mathcal{S}_k^-$, and rejected otherwise. By explicitly accounting for the remaining capacity of the routing plan, this policy aims to preserve flexibility for future requests.

4.5.3. Rollout acceptance policy

As previously explained, a rollout algorithm is a method that leverages a *base policy* to improve decision-making (Bertsekas et al., 1997). To do so, it evaluates different actions by *rolling out* the base policy from the current state to the end of the planning horizon. This approach ensures that each decision is made considering its long-term impact, leading to more effective and adaptive solutions. Due to the large number of simulations required, the base policy must be highly computationally efficient. In our case, the base policy corresponds to the greedy acceptance policy ϱ_G. Formally, given a routing policy ρ and a state $\mathcal{S}_k$, the rollout policy ϱ_R simulates the system over the set of scenarios Σ^k given that policy ϱ_G is in effect. That is, every time a new request appears, an acceptance decision is made using policy ϱ_G. Let $\phi^\sigma(\mathcal{S}_k^+)$ and $\phi^\sigma(\mathcal{S}_k^-)$ be the total reward obtained in scenario $\sigma \in \Sigma^k$ after accepting and rejecting request i_k, respectively. Then, request i_k is accepted if $\sum_{\sigma\in\Sigma^k} \phi^\sigma(\mathcal{S}_k^+) \geq \sum_{\sigma\in\Sigma^k} \phi^\sigma(\mathcal{S}_k^-)$. Otherwise, the request is rejected.

5. Perfect information bound

To further evaluate the quality of a policy, we establish a dual bound on the value of the optimal policy through the use of a perfect information

relaxation. To compute this bound we must solve the DPLRP exactly under the assumption that all the stochastic information is known and is available before the start of the planning horizon. This means that all the arrival times of the on-demand requests are known. To mathematically state this problem we start by defining the set Ω that contains all feasible park-and-loop routes. All the routes in this set satisfy the property that a request is never served before it is received. We also let a_{ir} be a binary parameter that takes the value 1 if route r serves request i, and 0 otherwise. Then, let y'_r be a binary variable that takes the value 1 if the park-and-loop route is selected, and 0 otherwise. Also, let z'_i be a binary variable that takes the value 1 if the on-demand request is served, and 0 otherwise. Then, the perfect information bound can be computed by solving the following mathematical integer program:

$$\max \sum_{i \in \mathcal{C}^d} z'_i \tag{23}$$

subject to

$$\sum_{r \in \Omega} a_{ir} y'_r = 1 \qquad \forall i \in \mathcal{C}^s \tag{24}$$

$$\sum_{r \in \Omega} a_{ir} y'_r = z'_i \qquad \forall i \in \mathcal{C}^d \tag{25}$$

$$\sum_{r \in \Omega} y'_r \leq |\mathcal{W}| \tag{26}$$

$$y'_r \in \{0, 1\} \qquad \forall r \in \Omega \tag{27}$$

$$z'_i \in \{0, 1\} \qquad \forall i \in \mathcal{C}^d. \tag{28}$$

The objective function (23) maximizes the number of on-demand requests served. Constraints (24) ensure that all scheduled requests are served. Constraints (25) state that on-demand requests can be served only once. They also guarantee that these requests can only be served by a route that visits their location. Constraints (26) impose an upper bound on the number of routes. Constraints (27)-(28) state that variables are binary. Formula-

tion (23)-(28) assumes that the set of all park-and-loop routes Ω is known. However, the size $|\Omega|$ of the route set grows exponentially with the number of requests. As an alternative, one can resort to column generation-based approaches. In this work, we use, with mild changes, the branch-price-and-cut algorithm introduced in Cabrera et al. (2023). We specifically adjusted the pricing problem to incorporate time windows at customer locations and redefined the formula for calculating the reduced cost of a route.

6. Computational experiments

In this section, we present our computational experiments. Our goal is to analyze the performance of the different online and offline policies. All algorithms and simulations were implemented in Java and executed on an Intel core i7 @2.30 GHz Quad-Core processor with 12 GB of RAM. We used Gurobi version 9.5.1 to solve linear and integer programs. The branch-price-and-cut algorithm to obtain the perfect information bound (PIB) was implemented in Java using the jORLib[1] library. We rely on CPLEX 20.1 to solve the master problem. All instances, the corresponding solution files, and a solution checker are available at `https://github.com/ncabrera10/DPLRP-SC`.

6.1. Test instances

We generated a total of 270 instances for this study based on the testbed designed by Zhang et al. (2023). They constructed a graph comprising 16,080 nodes representing the city of Vienna, Austria, which is publicly available. Each instance is denoted as $B_\delta_\lambda_\Lambda_n_\mathcal{W}$, where $B = \{1, 2, 3, 4, 5\}$ identifies the random seed, δ is the degree of dynamism, λ is the arrival rate of on-demand requests, Λ corresponds to the spatiotemporal distribution of the on-demand requests, n is the total number of requests, and $\mathcal{W}$ is the

[1]The latest version of jORLib can be downloaded at: http://coin-or.github.io/jorlib/.

number of workers. The degree of dynamism is a measure of the fraction of requests that are revealed dynamically during the planning horizon. Originally introduced by Lund et al. (1996), the degree of dynamism δ is defined as the ratio of new requests to the total number of requests. Using our notation, δ is mathematically defined as:

$$\delta = \frac{|\mathcal{C}^d|}{|\mathcal{C}^d| + |\mathcal{C}^s|} \in [0, 1] \,. \tag{29}$$

According to (29) the degree of dynamism increases if $|\mathcal{C}^d|$ increases or if $|\mathcal{C}^s|$ decreases. When the degree of dynamism is low, the routing plan requires infrequent updates, making the initial routing plan generated by offline policies particularly crucial. Conversely, when the degree of dynamism is high, the routing plan must be updated frequently, increasing the importance of effective acceptance and routing policies.

Table 2 details the parameters used. In Column 1, we list the number of requests, while Column 2 specifies the arrival rate in requests per minute. Column 3 indicates the degree of dynamism, and columns 4 and 5 show the number of on-demand and scheduled requests, respectively. In Column 6, we detail the number of workers. For each combination of parameters, we generated 15 instances. We calculated driving and walking times based on assumed speeds of 20 km/h and 4 km/h, respectively. The service time for each request ranges from 20 to 35 minutes, with a parking time set at 5 minutes. The maximum walking distance between two locations is 2.5 km, and each worker has a daily walking limit of 5 km. Each walking subtour lasts up to 2 hours, and the total planning horizon spans 7 hours. These parameters were selected based on a real-world application described in Cabrera et al. (2022).

The process to generate an instance proceeds as follows. First, the target number of on-demand requests is set to $\hat{N}^d = \lceil (f - e), \lambda \rceil$, where λ is the target arrival rate. Second, the number of scheduled requests is com-

Table 2: Instances information.

n	λ	δ	$\lvert\mathcal{C}^d\rvert$	$\lvert\mathcal{C}^s\rvert$	$\lvert\mathcal{W}\rvert$
30	0.007	0.10	3	27	3
30	0.018	0.25	6	24	3
30	0.036	0.50	15	15	3
30	0.054	0.75	24	6	3
30	0.064	0.90	27	3	3
30	0.068	0.95	29	1	3
50	0.012	0.10	5	45	6
50	0.030	0.25	11	39	6
50	0.060	0.50	25	25	6
50	0.089	0.75	39	11	6
50	0.107	0.90	45	5	6
50	0.113	0.95	47	3	6
100	0.024	0.10	10	90	12
100	0.060	0.25	25	75	12
100	0.119	0.50	50	50	12
100	0.179	0.75	75	25	12
100	0.214	0.90	90	10	12
100	0.226	0.95	95	5	12

puted as $|\mathcal{C}^s| = \left\lceil \hat{N}^d \cdot \frac{1-\delta}{\delta} \right\rceil$, where δ is the target degree of dynamism. We then randomly sample $|\mathcal{C}^s|$ locations from the graph to serve as scheduled requests, all assigned an arrival time of 0. Third, the arrival times of the on-demand requests are generated sequentially. Let $t_0 = e$. For each $i \in 1, \ldots, \hat{N}^d$, request i is generated only if $t_{i-1} \leq f - e$, with arrival time $t_i = t_{i-1} + \Delta t_i$, where $\Delta t_i \sim \text{Exponential}(\lambda)$. Fourth, service times are drawn independently for all requests. Finally, any on-demand request i that cannot be served within the planning horizon is discarded. The realized on-demand set $\mathcal{C}^d$ therefore satisfies $|\mathcal{C}^d| \leq \hat{N}^d$. The values reported in Table 2 reflect typical realized counts across the 15 replications performed for each parameter combination.

In this process, we consider three spatiotemporal distribution types: uniform time-independent (UTI), clustered time-independent (CTI), and clustered time-dependent (CTD). For the UTI distribution, on-demand request

locations are randomly sampled across the graph. In contrast, for the CTI distribution, requests are sampled with equal probability from two distinct clusters. Lastly, for the CTD distribution, the likelihood of selecting a node from the second cluster increases linearly over time, resulting in a higher concentration of on-demand requests in this cluster as the planning horizon approaches its end.

6.2. Assessing the performance of the scheduling policies

In this section, we assess the performance of the scheduling policies in the subset of instances with a high degree of dynamism $\delta = \{0.75, 0.9, 0.95\}$, where routing and acceptance policies play a more critical role. For all policies that use the MSH matheuristic, we set $Q = 1000$, $T_{rge}^{max} = 60$ seconds, and $T_{asm}^{max} = 10$ seconds. For all anticipatory policies, we set the number of scenarios $|\Sigma^k|$ at each decision epoch $k \in K$ to 50. To compute the perfect information bound we use the default parameters as described in Cabrera et al. (2023). To compare the scheduling policies we consider two metrics. First, the acceptance rate which is computed as the ratio between the number of on-demand requests served and the total number of on-demand requests received during the planning horizon. Second, the computational time required to select an action at each decision epoch.

The perfect information bound problem is at the intersection of two challenging combinatorial optimization problems: the team-orienteering problem and the park-and-loop routing problem with time windows. Hence, solving it can be a daunting task. Our preliminary experiments confirmed this intuition as the branch-price-and-cut algorithm was consistently able to solve all instances with 30 requests, but was only able to solve a few of the instances with 50 and 100 requests. For this reason, we first evaluate the scheduling policies in the subset of instances with 30 requests. Figure 3 presents a boxplot of the acceptance rates obtained for each scheduling policy within this subset of instances. Policies are sorted based on the average acceptance

rate. Scheduling policies using the routing policies ρ_{MI} and ρ_{MS} are colored in red and blue, respectively, while those using ρ_{PI} and ρ_{PS} are colored in yellow and green. Darker shades indicate policies that rely on the offline routing policy ϑ_M, whereas lighter shades correspond to policies using ϑ_P. Finally, acceptance policies ϱ_G, ϱ_R, and ϱ_{MK} are distinguished by different patterns inside the boxplots. We depict the average perfect information bound with a red line. On average, the perfect information solutions serve 85.3% of the on-demand requests. As this figure shows, the $\vartheta_P - \rho_{PS} - \varrho_{MK}$ and $\vartheta_P - \rho_{PS} - \varrho_R$ scheduling policies are the ones that find the closest results. They fail to serve 14.65% of the maximum attainable demand (i.e., $\frac{85.3\%-72.8\%}{85.3\%} = 14.65\%$). This figure indicates that the scheduling policies that use the offline policy ϑ_P tend to outperform their counterparts that use the offline policy ϑ_M by increasing the acceptance rate up to 4 percentage points.

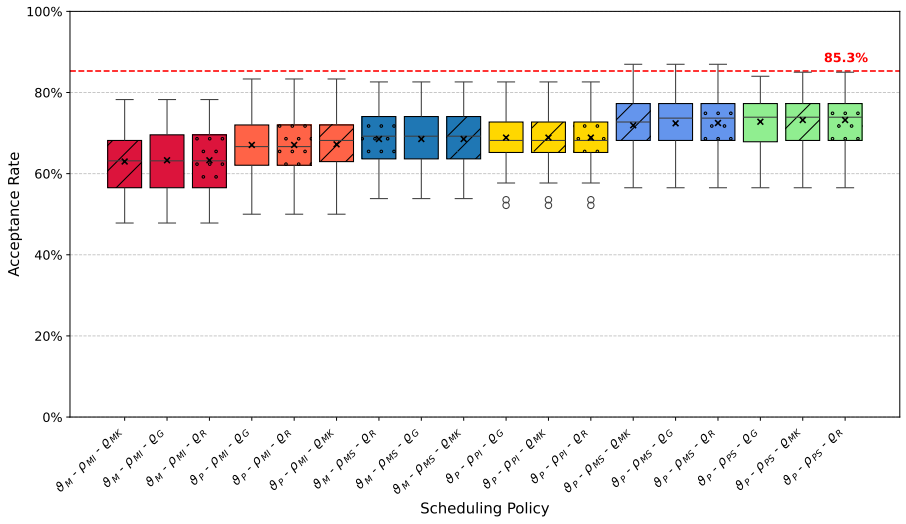


Figure 3: Comparison of scheduling policies acceptance rate and the perfect information bound in the subset of instances with 30 requests.

To further evaluate the performance of these scheduling policies, we also use the large instances with 50 and 100 requests. Figure 4 presents a boxplot of the acceptance rate across these instances, following the same color

convention as the previous figure. The results show that the $\vartheta_P - \rho_{PS} - \varrho_G$ scheduling policy achieves the highest acceptance rate at 77.5%. Our results indicate that considering the uncertainty of new request arrivals both in the design of the initial routing plan and when updating the routing plan positively impacts the acceptance rate. In contrast, the $\vartheta_M - \rho_{MI} - \varrho_{MK}$ policy, which neither anticipates new request arrivals when building the initial routing plan nor allows for reassigning requests among workers during updates, exhibits the lowest acceptance rate. This lack of flexibility often leads to difficulties in adapting to new requests.

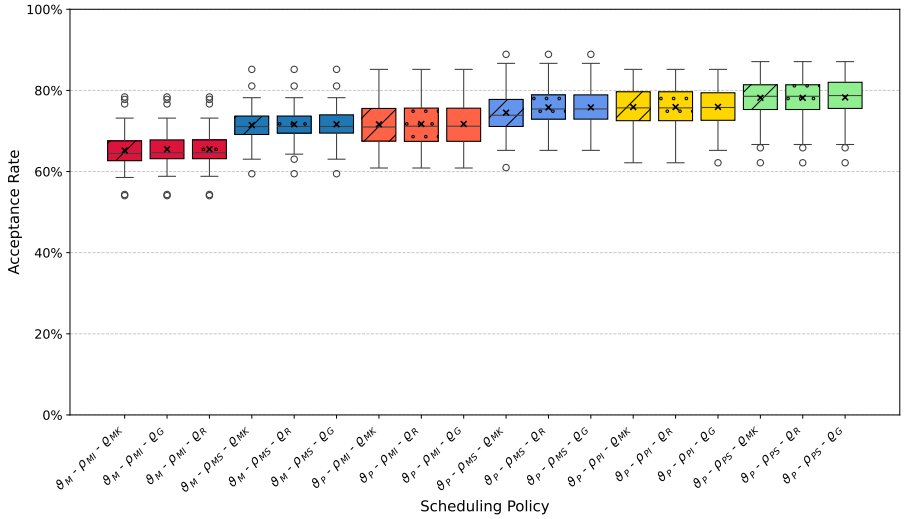


Figure 4: Comparison of scheduling policies acceptance rate in the subset of instances with 50 and 100 requests.

We also compare the scheduling policies based on the computational time required to make a decision when a new on-demand request appears. Figure 5 presents the average and maximum decision times, in seconds, across all instances. As shown, all policies make decisions in under one minute, regardless of the instance. However, scheduling policies $\vartheta_M - \rho_{MI} - \varrho_G$ and $\vartheta_P - \rho_{MI} - \varrho_G$ are the fastest. These two policies rely on the ρ_{PI} routing policy. In contrast, policies that use the ϱ_R acceptance policy tend to be slower on average, which is expected due to the high number of simulations

required before making a decision.

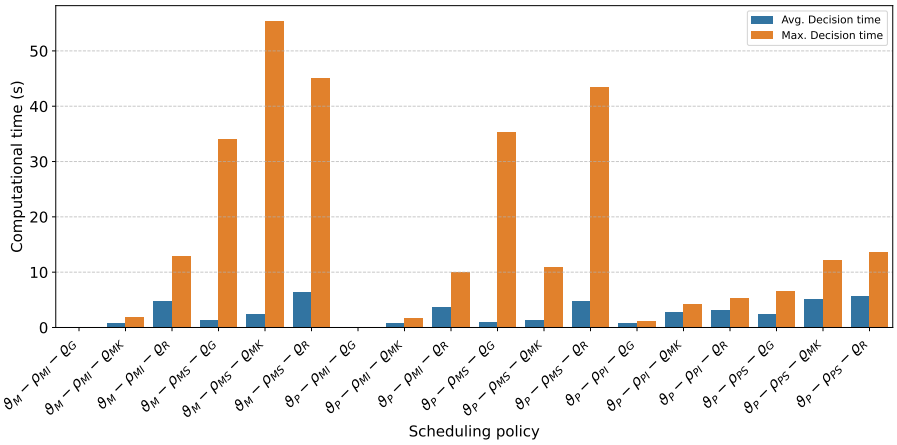


Figure 5: Comparison of scheduling policies computational effort in the subset of instances with 50 and 100 requests.

6.3. Assessing the performance of the offline policies

In this section, we analyze the performance of the offline policies as the degree of dynamism varies in the set of instances with 50 and 100 requests. Figure 6 presents a boxplot of the acceptance rate for each offline policy as a function of the degree of dynamism. The results indicate that policies using the potential-based offline policy consistently achieve a higher average acceptance rate, regardless of the degree of dynamism. However, the difference between using a non-anticipatory and an anticipatory policy becomes more pronounced when the degree of dynamism approaches 0.5, where the expected number of on-demand requests roughly equals the number of scheduled requests. This suggests that the choice between non-anticipatory and anticipatory policies should be carefully considered, especially when operating in scenarios with a moderate degree of dynamism.

To further explain the performance differences between the offline policies, we begin by showcasing the routing plans generated by each approach on a given instance. Figures 7a and 7b depict the initial routing plans created by the non-anticipatory and potential-based offline policies for instance

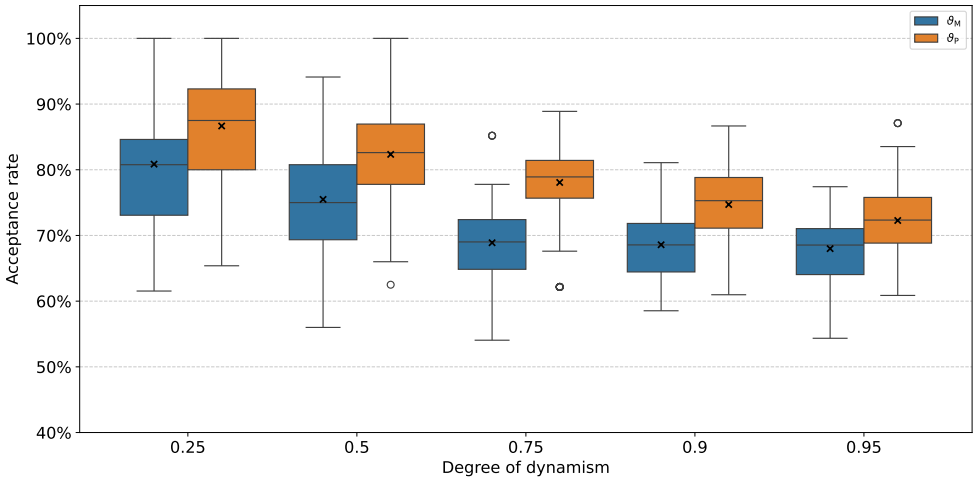
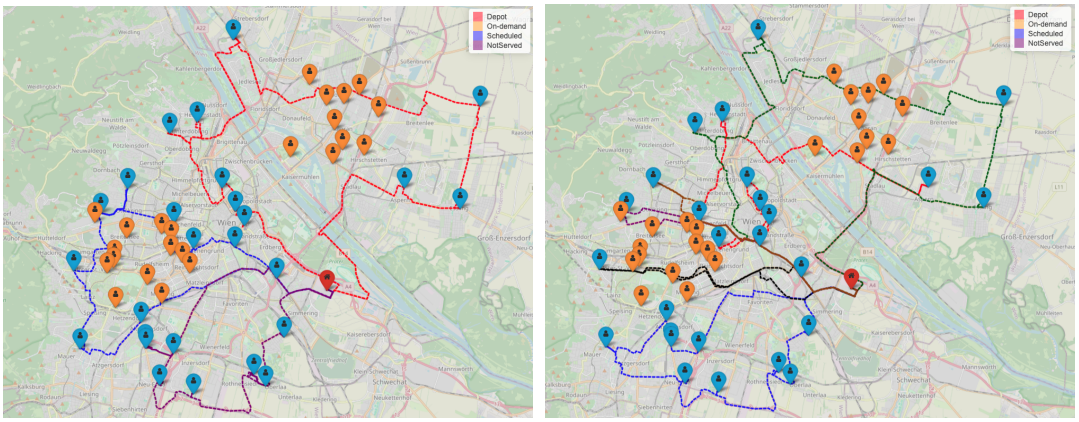


Figure 6: Comparison of the offline policies in the subset of instances with 50 and 100 requests.

1_0.5_0.06_CTD_50_6, which includes 25 scheduled requests and 25 on-demand requests, with six available workers. As these figures demonstrate, the initial routing plans from the two policies can differ significantly. The non-anticipatory policy, for instance, generates only three routes, leaving three workers idle at the depot. In contrast, the potential-based policy creates six routes, utilizing all available workers from the outset. Using these initial routing plans as inputs, we evaluated all scheduling policies and observed that the maximum number of served on-demand requests was 17 for the non-anticipatory plan and 20 for the potential-based plan. We believe that engaging all workers from the start provides greater flexibility in handling on-demand requests.

To further assess the impact of the offline policies, we compare the percentage of time that workers spend idle at the depot under each approach. Figure 8 shows the percentage of idle time as a function of the degree of dynamism. The results indicate that when the degree of dynamism is low, idle times are minimal, nearing 0%. However, as the degree of dynamism increases, idle times can rise to 34.6% and 29.7%. The figure also demonstrates that, regardless of the degree of dynamism, the offline potential-based policy



(a) Non-anticipatory (b) Potential based

Figure 7: Initial routing plans on instance 1_0.5_0.06_CTD_50_6.

generally reduces idle time. In some cases, the difference between the two policies is as much as 11.6%, highlighting the efficiency gains from using the potential-based approach.

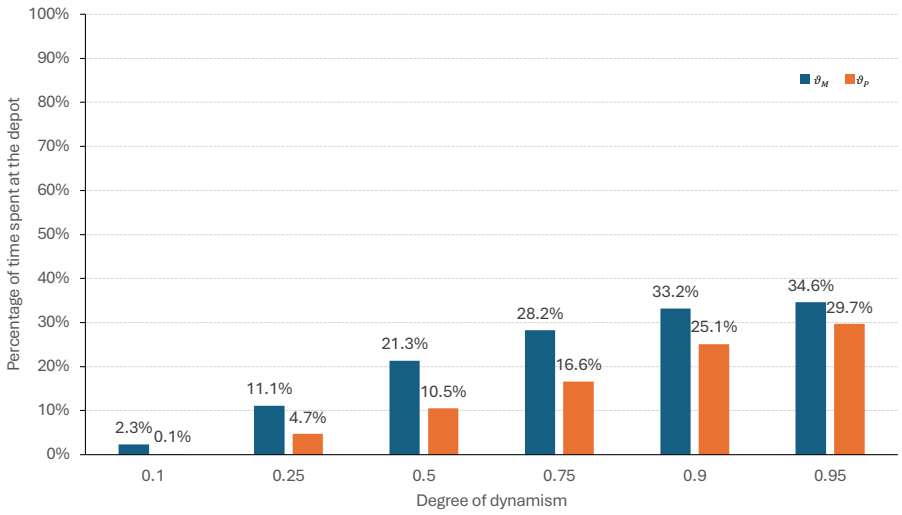


Figure 8: Comparison of the offline policies idle times in the subset of instances with 50 and 100 requests.

6.4. Assessing the performance of the online policies

In this section, we evaluate the performance of both the routing and the acceptance policies using the acceptance rate and the computational times

in the set of instances with 50 and 100 requests. Figure 9 presents a boxplot of the acceptance rate relative to the degree of dynamism. The results show that using the routing policy ρ_{PS} generally enhances the performance of the scheduling policies, with this improvement becoming more notable when the degree of dynamism exceeds 0.5. Specifically, when comparing routing policy ρ_{MS} with routing policy ρ_{PS}, we find that the potential-based routing policy increases the acceptance rate by an average of 3.75%. This positive impact is even more pronounced when comparing routing policies ρ_{MI} and ρ_{PI}, where the acceptance rate improves by an average of 5.55%. This increase can be attributed to the fact that these routing policies do not reassign requests between workers, making the anticipation of future requests more critical.

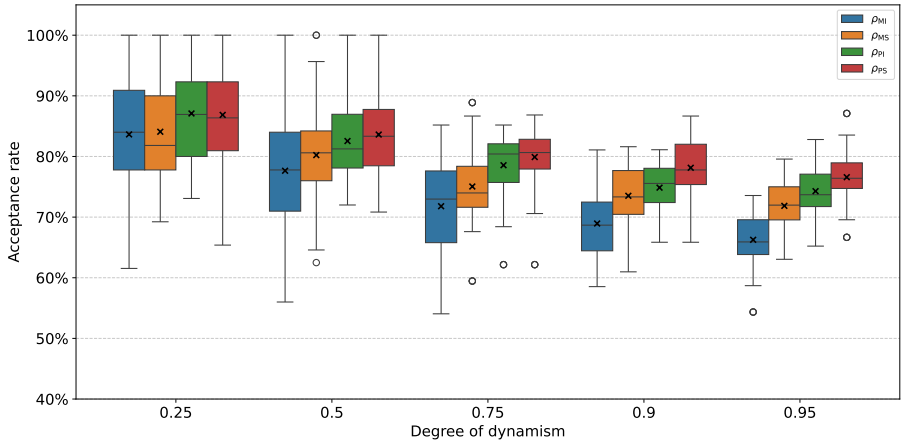


Figure 9: Comparison of the routing policies acceptance rate in the subset of instances with 50 and 100 requests.

We also compare the routing policies in terms of the computational time required to update the routing plan. Figure 10 shows both the average and maximum update times for each policy. As illustrated, routing policy ρ_{MI} consistently updates the routing plan in just a few milliseconds, making it the fastest on average, followed by routing policy ρ_{PI}. As mentioned earlier, these two policies do not reassign requests among workers, unlike policies ρ_{MS} and ρ_{PS}, which allows them to compute an updated routing plan much

faster. However, note that routing policy ρ_{PI} has an average update time of nearly 0.9 seconds, indicating that calculating the potential for each route in the pool and solving the set partitioning problem with the updated objective is significantly more computationally demanding. A similar observation can be made when comparing routing policies ρ_{MS} and ρ_{PS}, where the average update time increases from 1.13 seconds to 2.51 seconds.

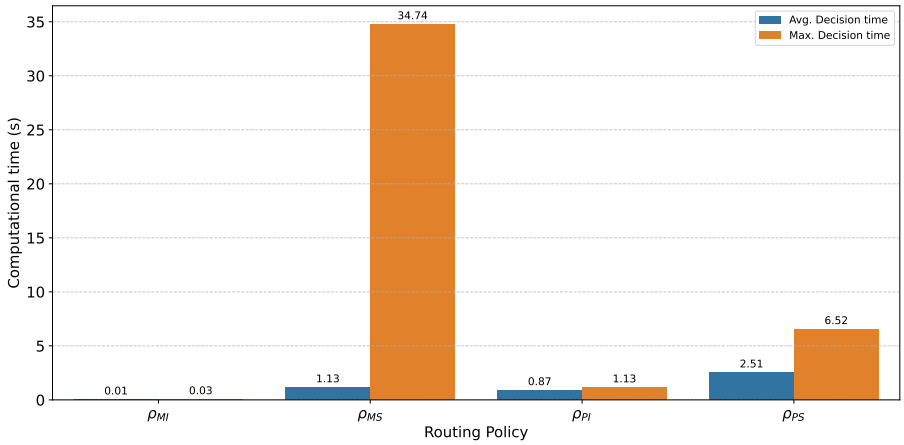


Figure 10: Comparison of the routing policies computational effort in the subset of instances with 50 and 100 requests.

Concerning the acceptance policies, Figure 11 presents a boxplot of the acceptance rate relative to the degree of dynamism when considering instances with 50 and 100 requests. As this figure shows, using an anticipatory acceptance policy only marginally improves the performance of the scheduling policies. A possible explanation for this behavior is that the anticipatory policies ϱ_R and ϱ_{MK} may not be exploring a broad enough set of future possibilities. This limited exploration could lead to suboptimal decision-making, particularly in a problem as complex as the DPLRP, where small changes in decision variables can have significant impacts. Additionally, the greedy policy may be already performing near optimally, leaving little room for the rollout policy to offer substantial improvements.

We also compare the acceptance policies in terms of the computational

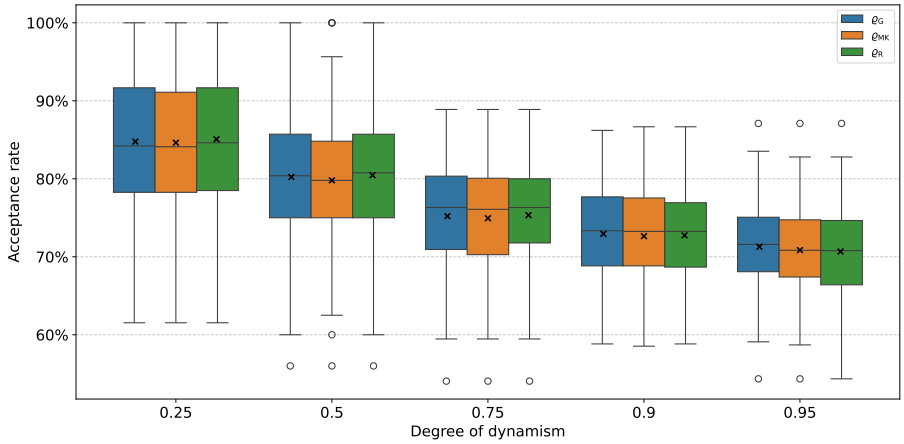


Figure 11: Comparison of the acceptance rates of different acceptance policies on the subset of instances with 50 and 100 requests.

time required to make an acceptance decision. Figure 12 shows the average and maximum decision times for each policy, when using the fastest routing policy, that is, ρ_{MI}. As this figure shows, the greedy policy ϱ_G can take a decision almost immediately. In contrast, the acceptance policies ϱ_{MK} and ϱ_R take almost 0.8 and 4.1 seconds on average to decide whether to accept or reject an on-demand request.

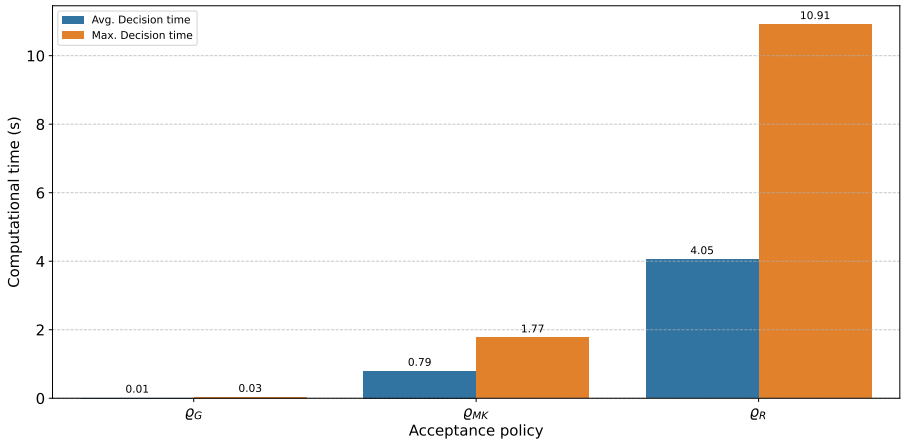


Figure 12: Comparison of the acceptance policies decision times in the subset of instances with 50 and 100 requests.

7. Concluding remarks

In this paper, we introduced the dynamic park-and-loop routing problem and proposed a route-based sequential decision process to address it. To solve this problem, we developed a set of scheduling policies that leverage the multi-space sampling heuristic. The use of MSH enables real-time consideration of complex routing decisions, including extending or shortening existing walking subtours, inserting or removing subtours, reassigning requests between workers, and reordering requests within a route. Each scheduling policy integrates an offline routing policy, an acceptance policy, and a real-time routing policy, ensuring adaptability to dynamic conditions. Additionally, we formulated the perfect information bound problem, providing a benchmark for evaluating the effectiveness of our approach.

We evaluated our set of scheduling policies using a newly generated set of instances with up to 100 requests. Our computational results demonstrate that these policies can produce solutions that, on average, are within 15% of the perfect information bound, showcasing their effectiveness under dynamic conditions. Regarding offline routing policies, our experiments indicate that employing an anticipatory policy can significantly enhance decision-making by anticipating sources of uncertainty. On the 30-request set, ϑ_P improves on ϑ_M by up to 4 percentage points; over the larger 50/100-request set, the improvement reaches 4.95% on average. Furthermore, our results highlight the superiority of the potential-based inter-route routing policy. This policy not only effectively anticipates the arrival of new requests but also facilitates the efficient reassignment of requests among workers, ensuring a more responsive routing plan. These findings underscore the importance of incorporating anticipation mechanisms in both offline and real-time routing decisions to improve overall system performance.

Future research should focus on incorporating other sources of uncertainty into the DPLRP such as stochastic travel times between locations

and stochastic service times. These factors could significantly impact the performance of the proposed routing policies, and accounting for them would enhance the robustness and adaptability of the solutions in real-world scenarios.

References

Belenguer, J., Benavent, E., Martinez, A., Prins, C., Prodhon, C., & Villegas, J. (2016). A branch-and-cut algorithm for the single truck and trailer routing problem with satellite depots. *Transportation Science*, *50*, 735–749.

Bent, R., & Van Hentenryck, P. (2007). Waiting and relocation strategies in online stochastic vehicle routing. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence (IJCAI-07)* (pp. 1816–1821). Hyderabad, India.

Bent, R. W., & Van Hentenryck, P. (2004). Scenario-based planning for partially dynamic vehicle routing with stochastic customers. *Operations Research*, *52*, 977–987.

Bertsekas, D. P., Tsitsiklis, J. N., & Wu, C. (1997). Rollout algorithms for combinatorial optimization. *Journal of Heuristics*, *3*, 245–262.

Cabrera, N., Cordeau, J.-F., & Mendoza, J. E. (2022). The doubly open park-and-loop routing problem. *Computers & Operations Research*, *143*, 105761.

Cabrera, N., Cordeau, J.-F., & Mendoza, J. E. (2023). Solving the park-and-loop routing problem by branch-price-and-cut. *Transportation Research Part C: Emerging Technologies*, *157*, 104369.

Chao, I.-M. (2002). A tabu search method for the truck and trailer routing problem. *Computers and Operations Research*, *29*, 33–51.

Coindreau, M.-A., Gallay, O., & Zufferey, N. (2019). Vehicle routing with transportable resources: Using carpooling and walking for on-site services. *European Journal of Operational Research*, *279*, 996–1010.

Derigs, U., Pullmann, M., & Vogel, U. (2013). Truck and trailer routing - problems, heuristics and computational experience. *Computers and Operations Research*, *40*, 536–546.

Ferrucci, F., Bock, S., & Gendreau, M. (2013). A pro-active real-time control approach for dynamic vehicle routing problems dealing with the delivery of urgent goods. *European Journal of Operational Research*, *225*, 130–141.

Gendreau, M., Guertin, F., Potvin, J.-Y., & Taillard, É. (1999). Parallel tabu search for real-time vehicle routing and dispatching. *Transportation Science*, *33*, 381–390.

Holguín-Veras, J., Leal, J. A., Sánchez-Diaz, I., Browne, M., & Wojtowicz, J. (2020). State of the art and practice of urban freight management: Part i: Infrastructure, vehicle-related, and traffic operations. *Transportation Research Part A: Policy and Practice*, *137*, 360–382.

Jodiawan, P., Côté, J.-F., & Coelho, L. C. (2025). The flexible park-and-loop routing problem. *Transportation Research Part C: Emerging Technologies*, *178*, 105173.

Le Colleter, T., Dumez, D., Lehuédé, F., & Péton, O. (2023). Small and large neighborhood search for the park-and-loop routing problem with parking selection. *European Journal of Operational Research*, *308*, 1233–1248.

Li, H., Wang, H., Chen, J., & Bai, M. (2020). Two-echelon vehicle routing

problem with time windows and mobile satellites. *Transportation Research Part B: Methodological*, *138*, 179–201.

Lund, K., Madsen, O. B., & Rygaard, J. M. (1996). *Vehicle routing problems with varying degrees of dynamism*. IMM, Institute of Mathematical Modelling, Technical University of Denmark.

Martinez-Sykora, A., McLeod, F., Lamas-Fernandez, C., Bektaş, T., Cherrett, T., & Allen, J. (2020). Optimised solutions to the last-mile delivery problem in london using a combination of walking and driving. *Annals of Operations Research*, *295*, 645–693.

Mendoza, J. E., & Villegas, J. G. (2013). A multi-space sampling heuristic for the vehicle routing problem with stochastic demands. *Optimization Letters*, *7*, 1503–1516.

Mitrović-Minić, S., & Laporte, G. (2004). Waiting strategies for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, *38*, 635–655.

Parragh, S. N., & Cordeau, J.-F. (2017). Branch-and-price and adaptive large neighborhood search for the truck and trailer routing problem with time windows. *Computers and Operations Research*, *83*, 28–44.

Pillac, V., Guéret, C., & Medaglia, A. L. (2012). *On the dynamic technician routing and scheduling problem*. Ph.D. thesis Ecole des Mines de Nantes.

Pillac, V., Guéret, C., & Medaglia, A.-L. (2018). A fast reoptimization approach for the dynamic technician routing and scheduling problem. *Recent Developments in Metaheuristics*, (pp. 347–367).

Powell, W. B. (2007). *Approximate Dynamic Programming: Solving the curses of dimensionality* volume 703. John Wiley & Sons.

Reed, S., Campbell, A. M., & Thomas, B. W. (2024). Does parking matter? the impact of parking time on last-mile delivery optimization. *Transportation Research Part E: Logistics and Transportation Review*, *181*, 103391.

Rothenbächer, A.-K., Drexl, M., & Irnich, S. (2018). Branch-and-price-and-cut for the truck-and-trailer routing problem with time windows. *Transportation Science*, *52*, 1174–1190.

Sarasola, B., Doerner, K. F., Schmid, V., & Alba, E. (2016). Variable neighborhood search for the stochastic and dynamic vehicle routing problem. *Annals of Operations Research*, *236*, 425–461.

Scherr, N., Reed, S., Campbell, A. M., & Thomas, B. W. (2026). Policies for the dynamic same-day delivery problem with walking and parking. *Transportation Research Part E: Logistics and Transportation Review*, *212*, 104898.

Schuijbroek, J., Hampshire, R. C., & Van Hoeve, W.-J. (2017). Inventory rebalancing and vehicle routing in bike sharing systems. *European Journal of Operational Research*, *257*, 992–1004.

Secomandi, N. (2001). A rollout policy for the vehicle routing problem with stochastic demands. *Operations Research*, *49*, 796–802.

Semet, F. (1995). A two-phase algorithm for the partial accessibility constrained vehicle routing problem. *Annals of Operations Research*, *61*, 45–65.

Shui, C., & Szeto, W. (2020). A review of bicycle-sharing service planning problems. *Transportation Research Part C: Emerging Technologies*, *117*, 102648.

Soeffker, N., Ulmer, M. W., & Mattfeld, D. C. (2024). Balancing resources for

dynamic vehicle routing with stochastic customer requests. *OR Spectrum*, (pp. 1–43).

Ulmer, M. W. (2017). *Approximate dynamic programming for dynamic vehicle routing* volume 61. Springer.

Ulmer, M. W., Goodson, J. C., Mattfeld, D. C., & Hennig, M. (2019). Offline–online approximate dynamic programming for dynamic vehicle routing with stochastic requests. *Transportation Science*, *53*, 185–202.

Ulmer, M. W., Goodson, J. C., Mattfeld, D. C., & Thomas, B. W. (2020). On modeling stochastic dynamic vehicle routing problems. *EURO Journal on Transportation and Logistics*, *9*, 100008.

Ulmer, M. W., Mattfeld, D. C., & Köster, F. (2018). Budgeting time for dynamic vehicle routing with stochastic customer requests. *Transportation Science*, *52*, 20–37.

Ulmer, M. W., & Streng, S. (2019). Same-day delivery with pickup stations and autonomous vehicles. *Computers & Operations Research*, *108*, 1–19.

Ulmer, M. W., & Thomas, B. W. (2018). Same-day delivery with heterogeneous fleets of drones and vehicles. *Networks*, *72*, 475–505.

Ulmer, M. W., & Thomas, B. W. (2020). Meso-parametric value function approximation for dynamic customer acceptances in delivery routing. *European Journal of Operational Research*, *285*, 183–195.

Villegas, J., Prins, C., Medaglia, A., & Velasco, N. (2013). A matheuristic for the truck and trailer routing problem. *European Journal of Operational Research*, *230*, 231–244.

Zhang, J., Luo, K., Florio, A. M., & Van Woensel, T. (2023). Solving large-scale dynamic vehicle routing problems with stochastic requests. *European Journal of Operational Research*, *306*, 596–614.

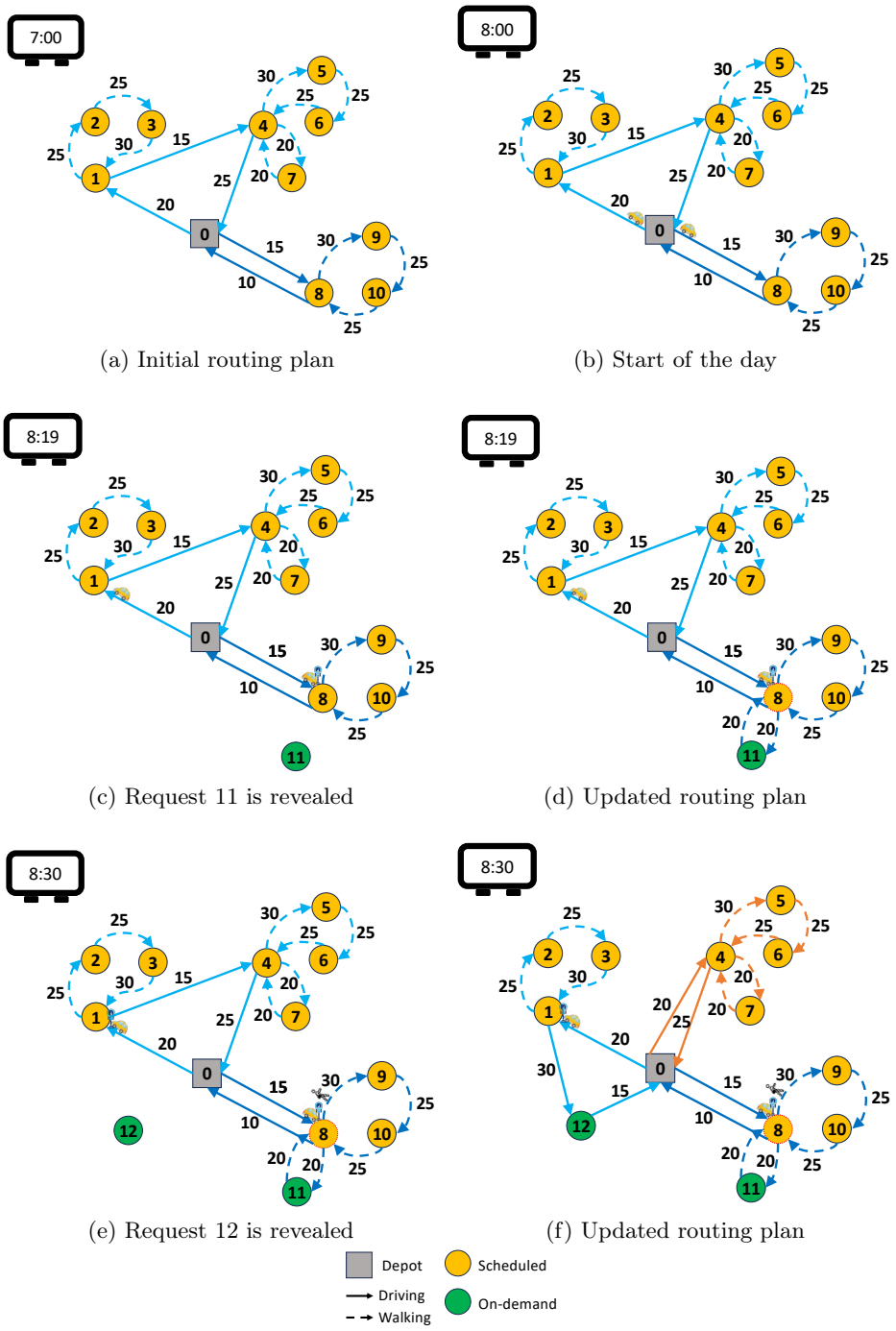
Zhang, J., & Van Woensel, T. (2023). Dynamic vehicle routing with random requests: A literature review. *International Journal of Production Economics*, *256*, 108751.

Zhou, H., Qin, H., Cheng, C., & Rousseau, L.-M. (2023). An exact algorithm for the two-echelon vehicle routing problem with drones. *Transportation Research Part B: Methodological*, *168*, 124–150.

Appendix A. Illustrative example of an instance of the DPLRP

Figure A.13 shows an example of an instance of the DPLRP with three workers. In this instance, the service time of every customer request is ten minutes and the time limit for a route is seven hours. Also, the parking time is set to two minutes. The driving or walking times between each location are indicated next to each arc. Figure A.13a shows the initial routing plan composed of two park-and-loop routes. This routing plan is built before the start of the workday. According to the initial routing plan, Worker 1 (light blue) departs at 8:00 am from the depot towards Request 1. There, the worker parks the vehicle before serving the request. Later, the worker walks to serve requests 2 and 3, before going back to retrieve the vehicle. Afterward, the worker drives to Request 4. After parking the car and serving the request, the worker walks to serve requests 5 and 6. Then, after going back to the parking spot, the worker walks to Request 7. Finally, after serving the request and retrieving the vehicle, the worker returns to the depot. Similarly, Worker 2 (dark blue) is scheduled to drive towards Request 8. Then, the worker walks to serve requests 9 and 10. Finally, after retrieving the vehicle, the worker drives to the depot.

Figure A.13b shows the park-and-loop routes being executed at the start of the planning horizon. Figure A.13c shows the park-and-loop routes being executed at 8:19 am, right after Request 11 appears. At 8:19 am, Worker 1 is about to reach the location of Request 1, while Worker 2 is already servicing Request 8. Due to the arrival of a new request and the proximity of Worker 2 to Request 11, the initial routing plan is modified as shown in Figure A.13d. In particular, a new subtour is added to the route of Worker 2 to ensure that Request 11 will be served. The route of Worker 1 remains unchanged. At 8:30 am, a new on-demand request is unveiled, namely, Request 12 as shown in Figure A.13e. To add this request to the routing plan, the route of Worker 1 is slightly changed and a new route is created (dark orange) as shown in



(a) Initial routing plan

(b) Start of the day

(c) Request 11 is revealed

(d) Updated routing plan

(e) Request 12 is revealed

(f) Updated routing plan

Figure A.13: Illustrative example of an instance of the DPLRP.

Figure A.13f. In addition, Worker 1 is now scheduled to serve Request 12.

This example illustrates the challenges and complexity of the DPLRP. When a new customer request appears, it is crucial to decide whether to accept or reject it. While accepting a new customer request may seem advantageous at the time, it could prevent the acceptance of two or more subsequent requests. Additionally, if the customer request is accepted, the routing plan must be adjusted to accommodate it. For instance, a subtour could be created within an existing route, or an existing subtour could be shortened or extended. Another option is to reassign requests among workers. This option could drastically change each route in the routing plan and could also involve creating new routes for idle workers at the depot. A solution method for the DPLRP must consider all these aspects and be able to quickly provide a solution.

Appendix B. Multi-space sampling heuristic example

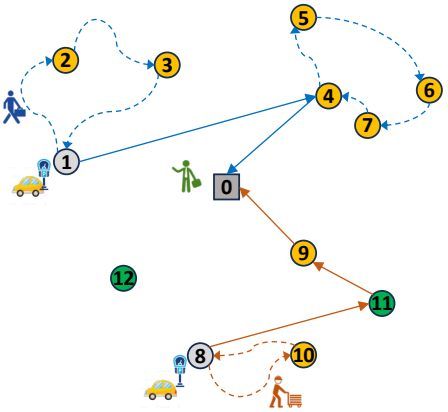
The purpose of this section is to show an example of how the multi-space sampling heuristic operates. Figure B.14 illustrates the two phases of MSH applied to an instance of the DPLRP with ten scheduled requests and two on-demand requests. Figure B.14a depicts the current state of the system: one worker, colored in green, is idle at the depot, while the other two workers, colored in blue and orange, are walking toward Requests 2 and 10, respectively. Figure B.14b shows a selection of routes generated during the first phase of the MSH, with each route beginning from the workers' current destinations or, in the case of the idle worker, their current location. Finally, Figure B.14c displays the updated routing plan. In this new routing plan, the worker in green is assigned to depart immediately toward Request 7.

Appendix C. Summary of scheduling policies

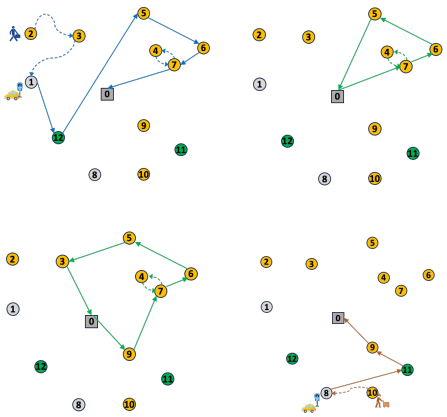
Table C.3 summarizes our set of scheduling policies. Each policy is a combination of an offline policy (ϑ_M, ϑ_P), a routing policy (ρ_{MS}, ρ_{MI}, ρ_{PS}, ρ_{PI}), and an acceptance policy ($\varrho_G, \varrho_{MK}, \varrho_R$). We denote a scheduling policy by $\vartheta - \rho - \varrho$.

Table C.3: Summary of the scheduling policies.

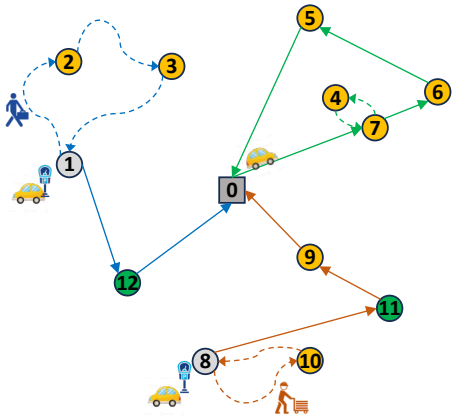
Scheduling policy	Offline policies		Routing policies				Acceptance policies		
	Non-anticipatory	Anticipatory	Non-anticipatory best-insertion	Non-anticipatory Inter-route	Anticipatory best-insertion	Anticipatory Inter-route	Greedy	Potential	Rollout
ϑ_M-ρ_{MI}-ϱ_G	X		X				X		
ϑ_M-ρ_{MS}-ϱ_G	X			X			X		
ϑ_M-ρ_{MI}-ϱ_{MK}	X		X					X	
ϑ_M-ρ_{MS}-ϱ_{MK}	X			X				X	
ϑ_M-ρ_{MI}-ϱ_R	X		X						X
ϑ_M-ρ_{MS}-ϱ_R	X			X					X
ϑ_P-ρ_{MI}-ϱ_G		X	X				X		
ϑ_P-ρ_{MS}-ϱ_G		X		X			X		
ϑ_P-ρ_{MI}-ϱ_{MK}		X	X					X	
ϑ_P-ρ_{MS}-ϱ_{MK}		X		X				X	
ϑ_P-ρ_{MI}-ϱ_R		X	X						X
ϑ_P-ρ_{MS}-ϱ_R		X		X					X
ϑ_P-ρ_{PI}-ϱ_G		X			X		X		
ϑ_P-ρ_{PS}-ϱ_G		X				X	X		
ϑ_P-ρ_{PI}-ϱ_{MK}		X			X			X	
ϑ_P-ρ_{PS}-ϱ_{MK}		X				X		X	
ϑ_P-ρ_{PI}-ϱ_R		X			X				X
ϑ_P-ρ_{PS}-ϱ_R		X				X			X



(a) Current routing plan



(b) Subset of feasible routes



(c) Updated routing plan

Figure B.14: Multi-space sampling heuristic example.