Computers & Operations Research 150 (2023) 106071

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

Survey in Operations Research and Management Science

Opportunities for reinforcement learning in stochastic dynamic vehicle routing

Florentin D. Hildebrandt [a], Barrett W. Thomas [b], Marlin W. Ulmer [a,*]

[a] Department of Management Science, Otto-von-Guericke-Universität, Magdeburg, Germany
[b] Department of Business Analytics, University of Iowa, Iowa City, United States

ARTICLE INFO

Keywords:
Stochastic dynamic vehicle routing
Reinforcement learning
Approximate dynamic programming
Mixed integer programming
Combinatorial optimization
Survey

ABSTRACT

There has been a paradigm-shift in urban logistic services in the last years; demand for real-time, instant mobility and delivery services grows. This poses new challenges to logistic service providers as the underlying stochastic dynamic vehicle routing problems (SDVRPs) require anticipatory real-time routing actions. The complexity of finding efficient routing actions is multiplied by the challenge of evaluating such actions with respect to their effectiveness given future dynamism and uncertainty. Reinforcement learning (RL) is a promising tool for evaluating actions but it is not designed for searching the complex and combinatorial action space. Thus, past work on RL for SDVRP has either restricted the action space, that is solving only subproblems by RL and everything else by established heuristics, or focused on problems that reduce to resource allocation problems. For solving real-world SDVRPs, new strategies are required that address the combined challenge of combinatorial, constrained action space and future uncertainty, but as our findings suggest, such strategies are essentially non-existing. Our survey paper shows that past work relied either on action-space restriction or avoided routing actions entirely and highlights opportunities for more holistic solutions.

1. Introduction

Operations research without mixed integer linear programming (MILP) might be just as unthinkable as computer science without machine learning. These iconic relationships are partly a product of the nineties, when computing became much cheaper, faster, and had larger memory storage. While machine learning's break-through took a further decade, the nineties mark the beginning of the golden age of mixed linear integer programming (MILP) for operations research. A new generation of exact MILP solvers, building on decades of theoretical work and advances in computer technologies, were suddenly capable of solving combinatorial optimization problems with millions of variables and thousands of constraints (Bixby, 2012). Planning problems like NP-hard vehicle routing problems (VRPs) were perfect applications for exact solvers because solution quality was pivotal, runtime secondary, and *static* environments were assumed.

Today, more real-world routing problems have become increasingly *dynamic*, and global interconnectedness, urbanization, ubiquitous information streams, and stronger service-orientation have only increased the dynamism and time-urgency of service fulfillment (see Archetti and Bertazzi (2021)). New services, like same-day or restaurant meal delivery, ride hailing, and emergency repair, force logistic service providers to anticipate future demand, adjust to real-time traffic information, or even incorporate unknown crowdsourced drivers to fulfill customer expectations (see Ulmer et al. (2021), Kullman et al. (2020), Ulmer and Thomas (2019) and Ulmer and Savelsbergh (2020)). Dynamism adds another dimension to vehicle routing optimization. The problem involves not only *searching* a large and constrained MILP-space for adequate routing, but also *evaluating* routing actions with respect to future stochastic dynamic developments. In this case, the methodology that so effectively solves static routing problems becomes only part of the solution as is does not solve the issue of evaluation. A problem class called stochastic dynamic vehicle routing problem (SDVRP) extends static VRP modeling to *dynamic* routing problems by accounting for the change of problem information over time and in ways that are only known in distributions. The optimization requires searching large and complex routing action spaces and to evaluate actions with respect to the future.

SDVRPs fall into the domain of dynamic combinatorial optimization, a sub-field of operations research, that addresses sequential decision problems with future uncertainty. The solution in sequential decision problems is a decision policy instead of a single solution vector. In theory, reinforcement learning (RL) is an ideal solution method for dynamic combinatorial optimization and SDVRPs because

* Corresponding author.
E-mail address: marlin.ulmer@ovgu.de (M.W. Ulmer).

https://doi.org/10.1016/j.cor.2022.106071
Received 3 March 2022; Received in revised form 28 October 2022; Accepted 30 October 2022
Available online 12 November 2022

(a) dynamic combinatorial optimization problems can naturally be modeled as Markov decision processes (MDP, see Powell (2019)) and
(b) learning architectures, such as neural networks (NN), can be trained offline to return fast and complex decision policies.

However, in SDVRPs the decision policy must assign a routing action to each state of the system. Such a routing action is usually represented as a sequence of stops for each vehicle that satisfies the specific problem constraints such as time windows, deadlines, pickup before delivery, or vehicle capacity (see Ulmer et al. (2020a)). RL methodology is challenged by such complex, often combinatorial, routing actions as it is more concerned with evaluating actions than searching complex action spaces. As such, the majority of available RL-techniques remains nearly untouched by past SDVRP-work. Instead, alternative strategies are applied, often based on the vehicle-routing community's well-known expertise using MILPs and focused more on efficiently finding an action given current information often at the cost of neglecting future uncertainty. RL methodology is employed, if at all, for problems that resemble resource allocation problems rather than routing problems or for subproblems of SDVRPs.

In this survey, we discuss the SDVRP class and its challenges. We analyze the literature on reinforcement learning for SDVRP originating from the operations research (based on the survey in Soeffker et al. (2022)) and the computer science community (focusing on the respective outlets). We show that a combined approach of searching a combinatorial action space while evaluating it via RL does not exist for dynamic routing problems. Either, the problems themselves are assignment problems without routing actions, or the combinatorial action space of the problem is restricted to a set of easily controllable actions. Based on the analysis, we provide guidance on how more holistic solutions could be achieved. The paper is structured as follows. In Section 2, we formally introduce SDVPRs, their components and model introducing two examples from the literature. In Section 3.1, we review the main concepts of RL and analyze how RL is applied to SDVRPs in the literature. Based on the analysis, we present an extensive future work section summarizing the most promising approaches in Section 4.

2. Stochastic dynamic vehicle routing problems

In this section, as they constitute the core of most SDVRPs, we give a short overview of static vehicle routing problems (VRPs) before we explain the intricacies of stochastic dynamic vehicle routing problems (SDVRPs).

In static (deterministic) VRPs the goal is to visit a set of spatially distributed locations N with a fleet of vehicles $\mathcal{V}$. All information is known at the time of decision making. There are many applications that induce static VRPs, among others are parcel or grocery delivery, pickup and delivery of (known) passengers or goods, or routing of a technician crew to repair or maintain equipment. In these cases, the work is planned in advance and the solutions are fixed during execution. In static VRPs, actions are usually *routing* focused , i.e., assignment of customers to vehicles and the sequencing of the vehicle stops. Routing actions must satisfy constraints like the well-known "sub-tour elimination constraints" and sometimes must satisfy application specific constraints like capacity constraints, time-window constraints, and precedence constraints. A common objective is to minimize cost (e.g., travel time or distance) or to maximize service (e.g., the number of customers served). Mathematically, we can model static VRPs via a set covering formulation of the form:

$$\begin{aligned} \text{minimize} \quad & \sum_{r\in\Omega} c_r \cdot x_r \\ \text{subject to} \quad & \sum_{r\in\Omega} a_{ir} x_r = 1, \qquad \forall i \in N \\ & x_r \in \{0,1\}, \qquad \forall r \in \Omega. \end{aligned} \tag{1}$$

Solutions are modeled by a vector x that indicates whether route $r \in \Omega$ is implemented from the set of feasible routes Ω. The routes are required to satisfy a set of linear constraints, which are implicitly enforced in the set Ω. The vector a counts how often each node $i \in N$ is visited by each route so that we can assert that each node is visited by exactly one route. The objective is a linear function given by a route-dependent cost or reward vector c and the solution x taken. In this case, we model the problem of minimizing cost when all customers have to be served, but the alternative, maximizing services by a given cost constraint, can be modeled very similarly. An alternative way of modeling VRPs is via an arc-based formulation where decision variables indicate the arcs used in the solution.

The set of feasible routes Ω and the corresponding number of solution variables grows combinatorially when static VRPs increase in size. Additionally, constraints are often complex and even finding a feasible solution is challenging. The operations research community has developed several effective solution mechanisms to tackle these NP-hard MILPs, recent approaches include among others Branch-and-cut (e.g., see Lysgaard et al. (2004)) and Branch-cut-and-price (e.g., see Baldacci et al. (2011) and Desaulniers et al. (2008)). All methodologies exploit structures in routing action variables and constraints to systematically search the vast solution space. For a general overview on VRPs and their solutions, we refer the reader to Toth and Vigo (2014) and Braekers et al. (2016).

With the paradigm shift in the logistic business models towards instant gratification and real-time fulfillment, models that capture stochasticity and dynamism are needed alongside established VRP models. In dynamic problems, routing actions are taken under incomplete information. Examples for SDVRPs include courier or ride hailing services (e.g., Ghiani et al. (2009) and Agatz et al. (2012)), restaurant meal or instant delivery (e.g., Yildiz and Savelsbergh (2019) and Voccia et al. (2019)), as well as emergency repair or healthcare services (e.g., Maxwell et al. (2010) and Schmid (2012)). There are many areas of uncertainty including in customers (e.g., demand volumes, new requests, choices, Bent and Van Hentenryck (2004) and Ulmer (2020)), the environment (e.g., travel times or parking, Schilde et al. (2014)), and increasingly also the fleet itself (e.g., crowdsourced drivers and their behavior, Ulmer and Savelsbergh (2020) and Arslan et al. (2019)).

The result are stochastic dynamic decision problems that are generally modeled via a Markov decision process (MDP, Ulmer et al. (2020a)). We summarize the elements of a Markov decision process in Fig. 1. An MDP models a problem as a sequence of states $S_k \in \mathcal{S}, k = 0, \dots, K$ that contain the information available for decision making. Based on the state information, state-dependent actions $x_k \in \mathcal{X}(S_k)$ from an action space $\mathcal{X}(S_k)$ are derived by a policy $X^\pi : S_k \mapsto x_k$. A reward function $R(S_k, x_k)$ assigns each state–action-tuple an immediate reward which may be used as feedback to refine the policy (reward values can be negative). Finally, revelations of stochastic information W_{k+1} combined with the current state–action-tuple (S_k, x_k) are mapped by a transition function $S^M(S_k, x_k, W_{k+1})$ towards the next state S_{k+1}. This entire process is repeated over a set of decision steps $k = 1, \dots, K$ in the problem time horizon. For SDVRPs, the MDPs typically take the following form:

States S_k: A state $S_k \in \mathcal{S}$ comprises all information needed to select an action, thus, information about the customers (locations, time-windows or deadlines, request type, demand volume, etc.), the fleet (position, fill levels, free capacity, etc.), and the environment (traffic or parking situation, etc.). Most notably, it also contains the current route plan $\bar{\Omega}_k$, i.e. the routes planned in the last decision (updated to the current point in time). In essence, a state of the MDP contains similar information as the MILP presented in Eq. (1) with a state-dependent set of nodes N_k and a set of feasible routes $\Omega(S_k)$.

Actions x_k: Actions $x_k \in \mathcal{X}(S_k) := \mathcal{P}\big(\Omega(S_k)\big)$, i.e., $x_k \subset \Omega(S_k)$ as $\mathcal{P}$ maps $\Omega(S_k)$ to its power set, update the routing of the fleet. Thus, the action space is equivalent to the solution space of the MILP presented

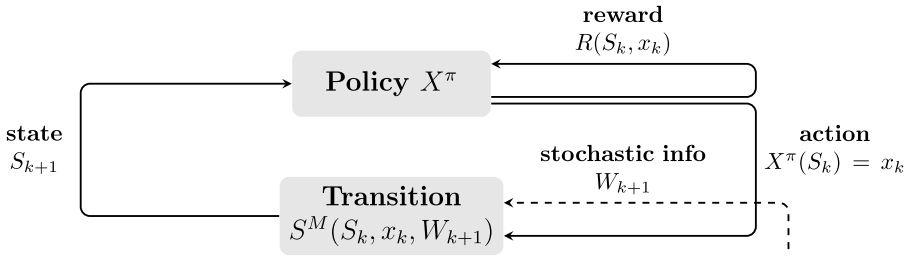

Fig. 1. Schematics of the Markov decision process.

in Eq. (1). However, routing actions are tentative, meaning that later stops in the routes might be altered in any future decision point; including the insertion of new requests into a route, the permutation of requests in a route, and the re-assignment of requests from one vehicle's route to another vehicle's route.

Reward function $R(S_k, x_k)$: The reward function evaluates the immediate impact of an action on the objective value and depends on the specific problem, e.g., routing cost, deadline violations, or increase in served customers. It is directly connected to the cost-function of the MILP in Eq. (1).

Stochastic information W_{k+1}: After an action x_k in S_k is taken, a realization of the stochastic information W_{k+1} is revealed. For SDVRPs, such information may change information about customers, fleet resources, or the environment the fleet operates in.

Transition function $S^M(S_k, x_k, W_{k+1})$: Based on state S_k, action x_k, and stochastic information W_{k+1}, the transition function leads to a new state S_{k+1} by updating the set of feasible routes $\Omega(S_{k+1})$, the set of planned routes $\bar{\Omega}_{k+1} \subset \Omega(S_{k+1})$, the set of requests N_{k+1}, etc.

Policies: A solution of the MDP is a policy $X^\pi \in \Pi$ that assigns an action $X^\pi(S_k) \subset \Omega(S_k)$ to every state S_k.

Objective: The objective of the MDP is to find an optimal policy X^{π^*} that maximizes the expected reward (or minimizes the expected costs) over the entire time horizon

$$\arg\max_{X^\pi \in \Pi} \mathbb{E}\left[\sum_{k=1}^{K} R\left(S_k, X^\pi(S_k)\right) \mid S_0\right]. \tag{2}$$

An important byproduct of the objective is the notion of a value function $V^\pi(S_k, x_k)$, also called state–action value function (or Q function). It depicts the expected reward that may arise in the future when taking action $x_k \subset \Omega(S_k)$ in state S_k and then following policy X^π. We denote the value function corresponding to the optimal decision policy as $V(S_k, x_k)$. Having access to the value function V assumes the possibility of optimal decision making for the rest of the process and of capturing all potential changes in information. The value function is not considered in the MILP in Eq. (1). Based on the value function, we can see that an optimal policy satisfies the Bellman Equation, maximizing the immediate reward plus the value of an action in a state and is given by:

$$X^{\pi^*}(S_k) = \arg\max_{x_k \in \Omega(S_K)} \{R(S_k, x_k) + V(S_k, x_k)\}. \tag{3}$$

When solving an SDVRP two components are particularly interesting: the action space and the value function. As previously discussed, searching the action space is analogous to finding a solution for the MILP in Eq. (1). However, in stark contrast to the static MILP, the values of the actions are not known.[1] Thus, solving SDVRPs combines two substantial challenges, *searching* an action in a complex, NP-hard mixed-integer program and *evaluating* each action with respect to future information changes and actions.

[1] Except in the very rare case when it is possible to solve the remaining tree of the MDP exactly, e.g., via recursion or by solving Eq. (3), which is almost never possible (see Powell (2019)).

2.1. Examples

The number of variants of SDVRPs is large, and each of them may lead to different model specifications. For example and as already discussed, problems may have different objectives (minimizing cost vs. maximizing revenue). Another important difference is the action space. Many problems have relatively complex action spaces where many routing constraints have to be satisfied such as time windows, capacity constraints, or precedence of stops for pickup and delivery. Other problems may not have any routing constraints at all and the action space is relatively simple. In the following, we introduce two extreme point examples in SDVRP: the stochastic dynamic pickup and delivery problem (SDPDP), motivated by ride sharing, and the stochastic dynamic assignment problem (SDAP), motivated by ride hailing. We will revisit both examples throughout this work to further explain various concepts. Both problems are strongly related as they are characterized by the dynamic request for transportation from an origin to a destination. Yet, as we highlight in the following, their problem complexities are vastly different.

Stochastic dynamic pickup and delivery problem (ride sharing)

Stochastic dynamic pickup and delivery problems (SDPDP) are motivated by applications such as ride sharing and courier services where passengers or goods have to be transported from origin to destination in a timely manner. In the following, we focus on ride sharing (see, e.g., Ulmer et al. (2020a)). In the ride sharing problem, a fleet of vehicles $v \in \mathcal{V}$ with passenger capacities $Q^v \in \mathbb{N}, \forall v \in \mathcal{V}$ serves requests for transportation from an origin to a destination. Requests are made during the service period and, if accepted, must be served before a given deadline. Request may be consolidated, e.g., we may pickup multiple requests before delivering them. The requests' deadlines are often far enough in the future to allow for such consolidations, which require construction of tentative routes. A tentative route is a planned sequence of requests served by a single vehicle (see Ulmer et al. (2020a) for an extensive discussion on tentative routes). While tentative routes might be changed entirely at a later decision point, the construction of tentative routes in every decision point is essential to assert that requests assigned to each vehicle can be feasibly served within the deadlines and without exceeding the vehicles' capacities. The Markov decision process (MDP) of a ride sharing problem may take the following form:

States S_k: States are (in our example and for ease of exposition) induced by equidistant time steps. Each state S_k is given by the current time t_k, the set of all planned routes $\bar{\Omega}_k$, the unserved requests given by their pickup locations P_k, dropoff locations D_k, number of passengers $q_{ik} \in \mathbb{N}$ for $i \in P_k$, and deadlines δ_{ik} for $i \in D_k$, and the travel time matrix tt. For ease of notation, we assume that for each pickup node $i \in P_k$, the corresponding delivery node is given by $i + n \in D_k$, where n is the number of requests and that it holds $q_{ik} = -q_{i+n,k}$. Thus, each request is given by $(i, i + n, q_{ik}, \delta_{ik})$. We denote the set of all pickup and delivery nodes as $N_k = P_k \cup D_k$. We further distinguish between newly revealed nodes N_k^{new}, given by $P_k^{\text{new}} := P_k \setminus P_{k-1}$ and

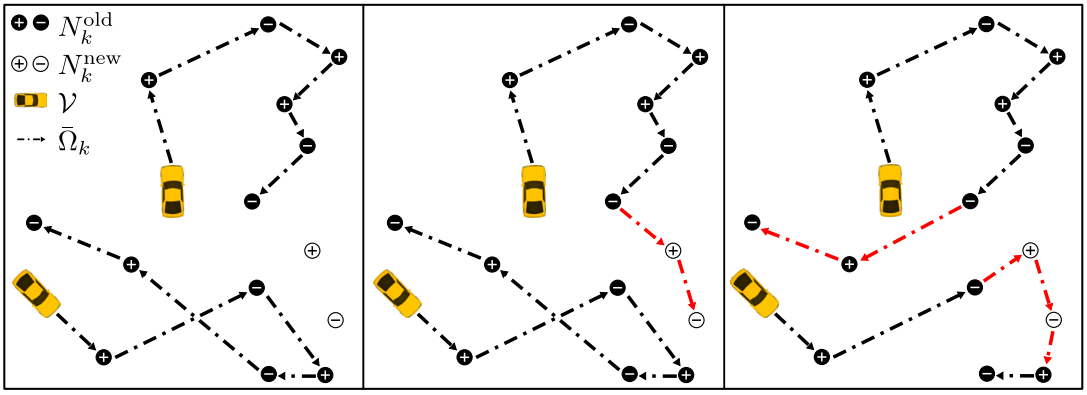

Fig. 2. Exemplary state (*left*) and two feasible actions (*center* and *right*) for a stochastic dynamic pickup and delivery problem.

$D_k^{\text{new}} := D_k \setminus D_{k-1}$, and previously revealed but still unserved nodes N_k^{old}, given by $P_k^{\text{old}} := P_k \setminus P_k^{\text{new}}$ and $D_k^{\text{old}} := D_k \setminus D_k^{\text{new}}$.

Actions x_k: An action $x_k \subset \Omega(S_k)$, i.e, $\mathcal{X}(S_k) := \mathcal{P}\big(\Omega(S_k)\big)$, is a set of feasible, tentative routes. The set is characterized by the following linear constraints with x_{ijk}^v being a binary variable that indicates whether vehicle $v \in \mathcal{V}$ drives from node i to node j in time step k

$$\sum_{v\in\mathcal{V}}\sum_{j\in N} x_{ijk}^v \leq 1 \qquad \forall i \in P_k^{\text{new}}, v \in \mathcal{V} \tag{4}$$

$$\sum_{v\in\mathcal{V}}\sum_{j\in N} x_{ijk}^v = 1 \qquad \forall i \in P_k^{\text{old}}, v \in \mathcal{V} \tag{5}$$

$$\sum_{j\in N} x_{ijk}^v - \sum_{j\in N} x_{i+n,j,k}^v = 0 \qquad \forall i \in P_k, v \in \mathcal{V} \tag{6}$$

$$\big(B_i^v + tt_{ij}\big)x_{ijk}^v \leq B_j^v \qquad \forall i,j \in N, v \in \mathcal{V} \tag{7}$$

$$\big(Q_i^v + q_{jk}\big)x_{ijk}^v \leq Q_j^v \qquad \forall i,j \in N, v \in \mathcal{V} \tag{8}$$

$$B_i^v + tt_{i,i+n} \leq B_{i+n}^v \qquad \forall i \in P_k, v \in \mathcal{V} \tag{9}$$

$$B_j^v \leq \delta_{jk} \qquad \forall j \in D_k, v \in \mathcal{V} \tag{10}$$

$$\max\{0, q_{ik}\} \leq Q_i^v \leq \min\{Q^v, Q^v + q_{ik}\} \qquad \forall i \in N, v \in \mathcal{V} \tag{11}$$

$$x_{ijk}^v \in \{0,1\} \qquad \forall i,j \in N, v \in \mathcal{V}. \tag{12}$$

Constraints (4) and (5) ensure that each node is visited at most once for new requests and exactly once for previously accepted requests. Constraints (6) assert that each pair of pickup and delivery nodes is served by the same vehicle. Constraints (7) and (8) enforce the consistency of the time variables B_i^v and load variables Q_i^v, $\forall k \in \mathcal{V}, i \in N$. Constraints (9) assert that each pickup node is visited before its corresponding delivery node and Constraints (10) introduce the service deadlines into the problem. Finally, Constraints (11) ensure that the passenger capacity Q^v of vehicle $v \in \mathcal{V}$ is not exceeded. We choose to present the action-space using an arc-based formulation, deviating from our set-based formulation in Problem (1), to better highlight the explicit constraints. Fig. 2 presents an exemplary state with two feasible actions. In the state (*left*), two vehicles (yellow), their tentative routes (dashed arrows) through previous requests (black circle), and a new, unassigned pickup- (white circle-plus) and delivery-request (white circle-minus) are shown. The center and right panel show different feasible integrations of the request into either vehicle's route for the given state. In the center panel, the request is appended to the route of the upper vehicle. In the right panel, the new request is inserted into the lower vehicle's route and a request from a previous decision point is re-assigned from the lower vehicle's route to the upper vehicle's route in order to satisfy all deadlines.

Reward function $R(S_k, x_k)$: The reward function is given by the number of newly accepted requests at each decision point

$$R(S_k, x_k) := \sum_{v\in\mathcal{V}}\sum_{i\in N_k}\sum_{j\in P_k^{\text{new}}} x_{ijk}^v. \tag{13}$$

In many ride sharing applications the costs of serving a request and the associated revenues are also included in the reward function but are ignored in this example for ease of exposition.

Stochastic information W_{k+1}: The stochastic information W_{k+1} contains the new requests N_k^{new}.

Transition function $S^M(S_k, x_k, W_{k+1})$: Based on state S_k, the chosen set of tentative routes x_k, and revealed stochastic information W_{k+1}, the transition function leads to a new state S_{k+1}

- by forwarding to the new point in time t_{k+1},
- by constructing the updated set of planned routes $\bar{\Omega}_{k+1}$ based on the action x_k and based on pruning nodes $(i, i+n) \in P_k \times D_k$ from x_k if the planned arrival at the delivery node is before the current time, i.e., $B_{i+n}^v \leq t_{k+1}$ for $v \in \mathcal{V}$ associated with the route that contains $(i, i+n)$,
- by updating the set of new customers N_k^{new}; and by updating the set of old but still unserved customers N_k^{old},
- and by constructing the updated set of feasible routes $\Omega(S_{k+1})$.

Policies X^π: A solution of the ride sharing problem is a policy $X^\pi \in \Pi$ that assigns a set of feasible, tentative routes $X^\pi(S_k) \subset \Omega(S_k)$ to every state S_k.

Objective: The objective of the ride hailing problem is to find an optimal policy X^{π^*} that maximizes the expected number of served requests over the entire time horizon

$$\arg\max_{X^\pi\in\Pi} \mathbb{E}\left[\sum_{k=1}^{K} R\left(S_k, X^\pi(S_k)\right) \mid S_0\right]. \tag{14}$$

Stochastic dynamic assignment problem (ride hailing)

Some stochastic dynamic vehicle "routing" problems do not require routing as they reduce to stochastic dynamic assignment problems (SDAP). Common examples are ride hailing and instant delivery services (see, e.g., Kullman et al. (2020)). In ride hailing, a fleet of vehicle serves requests for passenger transportation from an origin to a destination. Requests must be attended to within a very narrow deadline or they leave the system unserved. Constructing routes is not needed as the sharing of resources is prohibited and no service guarantee has to be given to customers. Thus, there is no need for tentative routes, and we model the fleet by a vector that specifies where and when each vehicle becomes available again. Each requests is modeled by its origin, destination, and deadline. Once a vehicle becomes available, we may assign it to serve a request from the set of active requests in the system or leave it idling if no request is available. The set of feasible requests for a specific vehicle is constructed by ensuring that the travel time from the vehicle's location to the request's location is smaller than

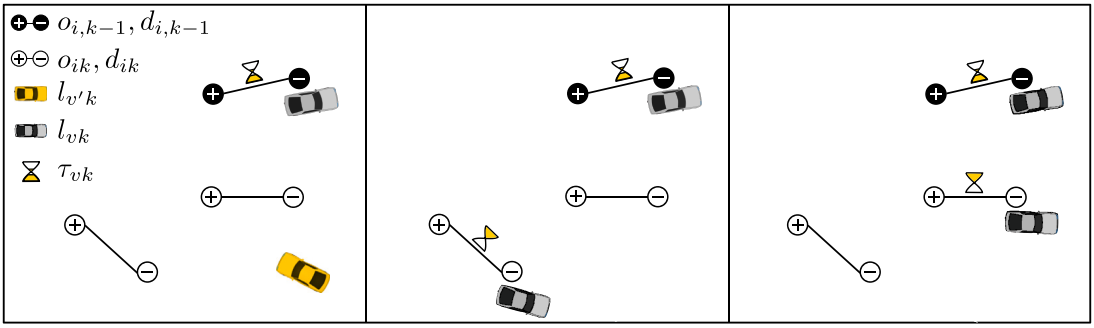

Fig. 3. Exemplary state (*left*) and two feasible actions (*center* and *right*) for a stochastic dynamic resource allocation problem.

the deadline. The corresponding Markov decision process (MDP) for a generic ride hailing problem could take the following form.

States S_k: The state S_k is given by the current time t_k, the scalar τ_{vk} specifying the time until each vehicle $v \in \mathcal{V}$ in the fleet becomes available again, the coordinate l_{vk} specifying where each vehicle becomes available again, all unassigned requests $i \in N_k$ given by their deadline δ_{ik} (remaining time to serve them), their origin and destination o_{ik}, d_{ik}, and a travel time matrix tt. We assume that states are induced when one vehicle $v' \in \mathcal{V}$ becomes available again (this is a simplification and it is generally sensible to let new customer requests also induce states).

Actions x_k: Let $v' \in \mathcal{V}$ denote the vehicle that became available, i.e. $\tau_{v'k} = 0$. Then, the action space may be modeled as the assignment problem

$$\sum_{i \in N_k} x_{ik}^{v'} \leq 1 \tag{15}$$

$$x_{ik}^{v'} \cdot tt(l_{v'k}, o_{ik}) \leq \delta_{ik} \quad \forall i \in N_k \tag{16}$$

$$x_{ik}^{v'} \in \{0,1\} \quad \forall i \in N_k. \tag{17}$$

Constraint (15) ensure that the available vehicle serves at most one request and Constraints (16) assert that the assigned request is picked up within the corresponding deadline. In summary, an action is given by a request $i \in N_k$ to assign to the free vehicle $v' \in \mathcal{V}$. We illustrate the action space for an exemplary state in Fig. 3. The state (*left*) consists of two vehicles and two new request. One vehicle (gray) is busy serving a previous request (black circles). Its remaining busy time is indicated by an hour glass. The yellow vehicle finished its last job and is now free to be assigned to one of the two requests. Either we assign it to the request in the left corner (*center*) or to the request in the center (*right*). The action we choose affects the fleet distribution and the busy time of vehicles in the next state and must therefore be carefully evaluated.

Reward function $R(S_k, x_k)$: The reward function is given by the number of newly assigned requests

$$R(S_k, x_k) = \sum_{i \in N} x_{ik}^{v'}. \tag{18}$$

Stochastic information W_{k+1}: The stochastic information W_{k+1} is given by the new set of requests N_{k+1}, which is the result of customer leaving the system as they have not been served before their deadline and new customers entering the system.

Transition function $S^M(S_k, x_k, W_{k+1})$: Based on state S_k, assignment action x_k, and stochastic information W_{k+1}, the transition function leads to a new state S_{k+1} by setting $t_{k+1} = t_k + \min_{v \in \mathcal{V}} \tau_{vk}$; updating the times at which vehicles become available by setting $\tau_{v'k} = tt(l_{v'k}, o_{ik}) + tt(o_{ik}, d_{ik})$ and then updating $\tau_{v,k+1} = \tau_{vk} - \min_{v' \in \mathcal{V}} \tau_{v'k}, \forall v \in \mathcal{V}$; by updating the location of the previously free vehicle $l_{v'k} = \sum_{i \in N_k} x_{ik}^{v'} d_{ik}$; by updating the requests' deadlines $\delta_{ik+1} = \delta_{ik} - \min_{v \in \mathcal{V}} \tau_{vk}$; and by constructing the new set of requests N_{k+1} according to the stochastic information W_{k+1}, by removing requests $i \in N_k$ with $\delta_{i,k+1} \leq t_{k+1}$, and by excluding the request assigned to vehicle v' by x_k.

Policies: A solution of the ride hailing problem is a policy $X^\pi \in \Pi$ that assigns a request $X^\pi(S_k) \in N_k$ to the free vehicle in every state S_k.

Objective: The objective of the MDP is to find an optimal policy X^{π^*} that maximizes the expected number of assigned requests over the entire time horizon

$$\arg\max_{X^\pi \in \Pi} \mathbb{E}\left[\sum_{k=1}^{K} R\left(S_k, X^\pi(S_k)\right) \mid S_0\right]. \tag{19}$$

2.2. A history of SDVRP methods

In this section, we briefly give an overview on the development of SDVRP-methods and illustrate their functionality based on the first example, the SDPDP. For a comprehensive discussion on SDVRPs, definitions of methodology, and the related literature, we refer to Soeffker et al. (2022).

While the earliest reference to the SDVRP seems to be the 1970s with additional work coming in the 1980s, there was a significant increase at the turn of the millennium (Psaraftis et al., 2016). Confronted with the new nature of stochasticity and dynamism, i.e., the need to search and evaluate complex action spaces, researchers initially divided in two schools of thought, focusing on one of the two needs.

A major part of early work focused on optimizing the resource usage given the current state information. To this end, authors turned to the well-known MILP-solution methods for thoroughly searching the action space, treating a state of the MDP as an isolated instance on a rolling-horizon fashion. Thus, in every state, the most efficient routing is determined based on the immediate cost or reward. Then, after observing new information, the routing is either updated or newly created, again based on the immediate cost or reward (see, e.g., Gendreau et al. (1999), Ichoua et al. (2000), or Branchini et al. (2009)). Given the SDPDP-example in Fig. 2, such a method may for example apply a tabu search on the previous routing solution to allow both the service of the new request and very time-efficient routes (Gendreau et al., 1999). The outcome might be the action in the right part of the figure. A major goal of this group of papers was to search the action space for efficient routing solutions very fast to allow real-time decision making. However, the evaluation of actions was limited to the immediate cost or reward, and, therefore the approach is "myopic" and ignores future realizations.

Another part of early work focused on identifying characteristics of valuable actions with respect to future dynamics, for example, routing solutions that ensured resources to stay flexible with respect to future changes. To this end, authors analyzed decision making in practice, idealized problem settings, as well as the SDVRPs' characteristics to handcraft fitting strategies such as waiting or threshold-strategies (e.g., Mitrović-Minić et al. (2004), Thomas (2007), or Pureza and Laporte (2008)). Thus, while not directly evaluating actions with respect to their future value, actions were selected that were likely to produce

flexible states and consequently better future values. For the SDPDP-example in Fig. 2, such a policy may assign new requests preferably to the vehicle with smallest workload to keep the workload balanced and no vehicle overly congested. The outcome might be the action in the central part of the figure. Such *policy function approximations* (PFAs, compare (Powell, 2019)) generally ignore the vast number of potential actions as well as costs and value function. That is, they look at a very limited subspace of the action space, a procedure we call *action-space restriction*.

Recently, there have been approaches to combine the two types of approaches, integrating practical or analytical insights in static optimization procedures. Such methods are summarized under the term *cost function approximation* (CFA). The main idea of a CFA is to integrate the general concept of a PFA into the MILP-model of a state. This allows an extensive and fast search of the action space while also evaluating actions with respect to their future value (at least, implicitly). To this end, past works manipulated the MILP to incentivize flexibility or avoid myopic decision making, for example, via safety buffers or penalty terms (see e.g., Chen et al. (2018), Ulmer et al. (2020b), or (Riley et al., 2020)). For a problem similar to the SDPDP-example in Fig. 2, Ulmer et al. (2021) added an artificial parameter b to the deadline constraints (10) of the action space:

$$B_j^v - b \leq \delta_{jk} \; \forall j \in D_k, v \in \mathcal{V}.$$

This parameter b controls the balance between consolidation and flexibility. With a small b, customers might be consolidated to a vehicle as it leads to the most efficient routes (Fig. 2, right) while with increasing b, customers have to be served earlier and the workload is distributed over the vehicles more evenly leading to a potentially more flexible fleet setup (Fig. 2, center).

CFAs can theoretically ensure a thorough search and an accurate evaluation of the action space. However, it is unclear where and how the corresponding MILP-models should be manipulated. Also, most of the CFAs turn to static state-independent parametrization and still require extensive tuning. Further, Ulmer et al. (2021) showed that a fitting parametrization could be rendered ineffective by slight changes in the instances. In essence, the success of CFAs is still limited at the moment. Given their theoretical advantages, we see CFAs as one of the more promising approaches though, especially when combined with RL. We will discuss potential approaches in Section 4.

While PFAs and CFAs evaluate decisions analytically, an alternative, data-driven stream of methodology has developed simultaneously. Such methods use samples of stochastic information to predict future developments. One type of methodology is reinforcement learning, which we discuss in the subsequent section in detail. The other methodology samples information *online* in each state, either as scenarios or via simulation. Notably, the differentiation between search- and evaluation-focused methodology holds for data-driven methods as well.

Scenario-based methods sample several sets of information realizations (e.g., requests). Then, a solution is sought that performs well over all scenarios. In rare cases, this is done via stochastic programming (e.g, Sáez et al. (2008), Ghiani et al. (2009) and Klapp et al. (2018)) where a solution is found considering the different scenarios directly. However, due to the complex decision making within the scenarios and the limited runtime, scenarios are often solved individually. Since they are deterministic, the solution are usually derived via the strong static VRP methodology, e.g., via mixed-integer programming (e.g., Song et al. (2020)) or metaheuristics (e.g., Schilde et al. (2014)). Then, a *consensus* solution is used to identify the solution "most similar" to the others. The solution cleaned from the sampled information is then used as the action in the state. This general concept is known as the multiple-scenario approach (MSA), initially introduced by Bent and Van Hentenryck (2004) (for a recent overview, see Ausseil et al. (2022)). For the SDPDP example in Fig. 2, an MSA might sample future demand scenarios, i.e., sets of future requests. Then, it would solve the individual deterministic scenarios, maximizing the number of served requests. Finally, it would compare the assignment and routing decisions among the scenario solution and select the solution most similar to the others and implement it. MSAs and related methods do not evaluate actions via the Bellman equation, but rather use the scenarios to determine a solution that accounts for the future. However, the search of the action space is comprehensive.

The second type of sampling methodology does exactly the opposite, a limited search of the action space and a thorough evaluation of the limited actions considered. These methods restrict the action space, e.g., via decomposition. The remaining potential actions are then evaluated by simulating several trajectories of future information realization and decision making. The decision making within each simulation is generally done via a runtime-efficient heuristic. Each trajectory provides a realized value. The average value for each potential action is then used in Eq. (3) to find the best action in the state. Such methods are known in the OR-community under the term of *post-decision rollout algorithm* (RA, see Goodson et al. (2017) for a classification and Secomandi (2001), Ulmer et al. (2019)) and in the computer science community as *Monte Carlo tree search*. For the SDPDP example in Fig. 2, an RA may decompose decision making into assignment and routing. Routing would be done by a simple insertion strategy. Thus, three potential assignment decisions (first vehicle, second vehicle, no service) would provide three potential actions to evaluate. They would be evaluated by simulating future requests and future decision making where both routing and assignment is determined via a runtime-efficient heuristic. From the three potential actions, the one with the maximum average number of services would be selected. Due to the tremendous calculation effort for evaluating an action, RAs only work if the number of considered actions is very limited. Still, they have the major advantage that these actions are evaluated in detail and thoroughly with respect to the Bellman equation (3).

3. Solving SDVRPs with RL

In this section, we provide a short tutorial on two important RL methods before we analyze previous works that use RL to solve SD-VRPs.

3.1. Reinforcement learning

We provide a brief introduction on the most relevant reinforcement learning (RL) methods for SDVRP. For an extensive introduction into the topic, we refer the reader to Sutton and Barto (2018). In reinforcement learning, an agent learns a *policy* by interacting with a partially observable *environment*. The agent aims to maximize delayed *rewards* either by approximating the *value function* according to the Bellman equation (3) or by directly learning the "right" actions for a state. In short, RL is learning the long-term value of actions, either explicitly in the case of *value function approximation* or implicitly in the case of *policy gradient methods*, as we explain in the following.

3.1.1. Value function approximation

In Section 2, we stated that having access to the value function V enables us to derive an optimal decision policy by choosing the actions with maximal long-term rewards. Unfortunately, V is unknown. Value function approximation (VFA), also referred to as Q-learning, aims to approximate the unknown value function V using an estimator $\hat{V}$, as we highlight in Fig. 4.

The lower part of the figure ("action", "state", "transition") is identical to the MDP summarized in Fig. 1. This figure is now extended by a reinforcement learning agent (light gray box, "Agent") that is based on a value function approximation. The VFA $\hat{V}_k$ (dark gray box, "VFA") encodes the current knowledge by assigning each action in a state an expected long-term reward. The expected long-term rewards assigned to an action by the VFA are statistically learned based on past observations or more likely simulations of state–action-reward

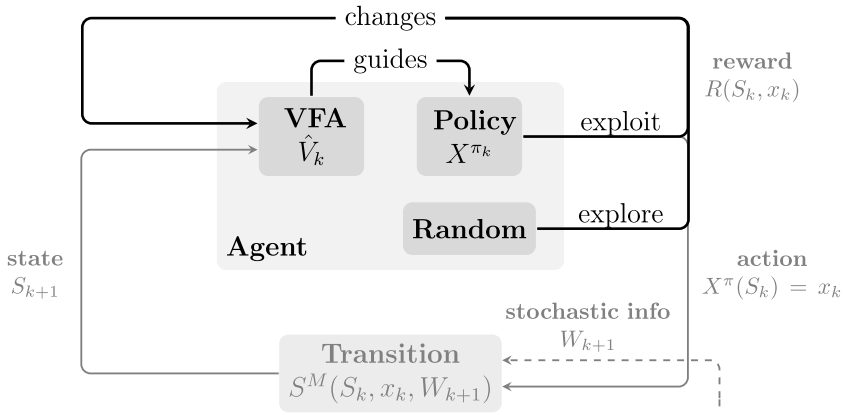

Fig. 4. Integration of a value function approximation in the MDP.

tuples ("changes"). In any state, we may exploit ("exploit") our current knowledge by choosing the "best" action x_k via the policy X^{π_k} (dark gray box, "Policy") guided by the VFA $\hat{V}_k$ ("guides"), i.e.,

$$X^{\pi_k}(S_k) = \underset{x \in \mathcal{X}(S_k)}{\arg\max}\, \hat{V}_k(S_k, x). \tag{20}$$

Instead of exploiting the value function, we may improve our knowledge by exploring ("explore") unknown actions, e.g., by choosing a random action (dark gray box, "Random"). In summary, in each state the agent either takes the best action according to Eq. (20) or chooses a random action. Putting theory into practice, a first challenge arises when choosing the approximation architecture that emulates V. Historically, the approximations may have been stored in non-parametric lookup tables (see e.g., George et al. (2008) and Ulmer et al. (2018a)) or as simple parametric functions (see e.g., Rivera and Mes (2017) and Ulmer and Thomas (2020)). This requires a coarse aggregation of the state space to a few meaningful features in order to reduce memory space or number of parameters in the model. Modern implementations use neural networks (see e.g., Kullman et al. (2020) and Chen et al. (2022)) as they are able to interpolate between known states and allow for finer state details, because they can operate directly on unaggregated states. Independent of the chosen approximation architecture, the value function approximation indirectly defines the agent's current policy X^{π}, as defined in Eq. (20). Therefore, most value function approximations assume that all actions in a state can be evaluated by enumeration; a strong limitation with regards to SDVRPs' combinatorial action spaces.

3.1.2. *Policy gradient methods*

Instead of implicitly learning to choose actions in each state by explicitly learning the value of actions via VFA, we may learn to explicitly choose actions in each state by implicitly learning their values. In such an approach, we interpret a policy X^{π} not as a mapping $S_k \mapsto x_k$ but as a probability distribution $\pi(\cdot|S_k)$ over the action space $\mathcal{X}(S_k)$. Based on this understanding of a policy, we may sample from the distribution $\pi(\cdot|S_k)$ to get an action $x_k \sim \pi(\cdot|S_k)$ in a state S_k. Typically, the probability distribution π is represented by a neural network $\pi(\cdot|S_k, \theta)$ which is parameterized by a set of weights θ and which takes the current state S_k as input. When employing the neural network $\pi(\cdot|S_k, \theta)$, actions are either selected by sampling according to $\pi(\cdot|S_k, \theta)$ or by deterministically choosing the action with the highest probability $x_k = \arg\max_{x \in \mathcal{X}} \pi(x|S_k, \theta)$.

Of course, the distribution $\pi(\cdot|\cdot, \theta)$ with its initial, untrained parameters θ might constitute a poor policy at the beginning. Thus, we want to improve its parametrization θ based on observations of states, actions, and rewards $S_k, x_k, R(S_k, x_k)$. When shaping θ to improve the probability distribution $\pi(\cdot|S_k, \theta)$, we follow a common principle of policy gradient methods. Actions that lead to high long-term rewards should have a higher probability than less rewarding ones. Thus, we modify the distribution $\pi(\cdot|S_k, \theta)$ by reinforcing (i.e., increasing) probabilities $\pi(x_k|S_k, \theta)$ associated with actions $x_k \in \mathcal{X}(S_k)$ that have led to high long-term rewards $v_k = \sum_{t=k+1} R(S_t, x_t)$ in a past simulation. In turn, we decrease the probabilities of actions if they have led to low long-term rewards for the given state. We explain this idea in technical terms in the following.

As discussed in Section 2, we describe the expected cost when following our policy X^{π_θ} from the current state S_k onward by $V^{\pi_\theta}\left(S_k, X^{\pi_\theta}(S_k)\right)$. In theory, we want to shape θ to increase the expected long-term reward $V^{\pi_\theta}\left(S_k, X^{\pi_\theta}(S_k)\right)$ associated with our policy X^{π_θ}. In practice, updating the policy's parameter θ to increase its associated long-term reward V^{π_θ} in each state is difficult as no explicit gradient of $V^{\pi_\theta}\left(S_k, X^{\pi_\theta}(S_k)\right)$ can be computed with respect to θ and given a state S_k and action x_k. In a common workaround, practitioners resort to empirical approximation of the gradient of the policy's value function based on our observed future cumulative rewards v_k:

$$\nabla_\theta V^{\pi_\theta}\left(S_k, X^{\pi_\theta}(S_k)\right) = \mathbb{E}\left[v_k \nabla_\theta \ln \pi(x_k|S_k, \theta)\right]. \tag{21}$$

This approach leads to the famous *REINFORCE* algorithm (Williams, 1992) summarized in Algorithm 1. In the *REINFORCE* algorithm, we generate $K+1$ state–action-reward-tuples by simulating an entire episode following our current policy $\pi(\cdot|\cdot, \theta)$. For each decision point $k = 0, \dots, K-1$, we compute the observed cumulative future reward v_k. We use this information to update the policy's parameters θ based on a step size α, a discount factor γ, our observed cumulative reward v_k, and the gradient of our policy π with respect to its parameters θ and given the corresponding state–action-tuple (S_k, x_k).

Input: A differentiable policy parameterization $\pi(\cdot|\cdot, \theta)$ with initial weights θ, stepsize $\alpha > 0$, discount factor $0 < \gamma \leq 1$.
Output: Trained policy parameters θ.
while *True* **do**
  Generate an episode $S_0, x_0, R(S_0, x_0), \dots, S_K, x_K, R(S_K, x_K)$, following $\pi(\cdot|\cdot, \theta)$
  for $k = 0, \dots, K-1$ **do**
    $v_k = \sum_{t=k+1} \gamma^{t-k-1} R(S_t, x_t)$
    $\theta \leftarrow \theta + \alpha \gamma^k v_k \nabla \ln \pi(x_k|S_k, \theta)$
  end
end

Algorithm 1: REINFORCE Algorithm.

A variety of different policy gradient methods originate from the REINFORCE algorithm. They all built on the observation that while Approximation (21) is true to the policy's value functions actual gradient in expectation (see Eq. (21)), it might be unreliable if the previously observed long-term rewards v_k have high variance. Thus, a common

remedy to reduce variance (while preserving bias) is to subtract a baseline from v_k. In one prominent class of policy gradient methods, referred to as actor–critic methods, a VFA is used as a baseline, which is effectively a combination of VFA and policy gradient method approaches (see Sutton and Barto (2018)). Independent of the baseline used, policy gradient methods can be applied to continuous action spaces even though they are not enumerable as the policy is explicitly learned. However, just like value function approximations, policy gradient methods fail for discrete action spaces when the number of actions presented in a state is too large. Additionally, it is difficult to enforce that the image, i.e., the set of all output values, of the policy gradient method (from an algorithmic perspective) coincides with the action space (from a model perspective) when constraints are complex as is the case for SDVRPs.

3.2. Past works on RL for SDVRPs

Having introduced the basics of reinforcement learning, we now turn towards its application to SDVRP. In most RL applications, the key is to evaluate actions in a given state. Searching the action space is typically not necessary because most instantiations of RL require that we are able to enumerate an action space, e.g., in Q-learning. Famous RL implementations for games such as Chess (Silver et al., 2017), Go (Silver et al., 2016), or Atari games (Mnih et al., 2015) are characterized by relatively small action spaces in each state despite the games' well-known complexity; e.g., in the Atari game "Breakout" we have only three possible actions ("left", "right", "do nothing") in each state. SDVRPs can have complex, combinatorial action spaces for which it might be difficult to construct a feasible action, let alone to enumerate all actions for a single state. For example, in ride sharing problems, we must construct tentative routes in each state that satisfy the requesting customers' deadlines as well as capacity constraints, precedence constraints, and travel time constraints. Such an action space grows combinatorially in the number of vehicles and requesting customers and is further complicated by the complex constraints. For that reason, past work on RL for SDVRP has either focused on assignment-based SDVRPs, as given in the ride hailing example of Section 2.1, or applied RL on manageable subproblems of route-based SDVRPs, as given in the ride sharing example in Section 2.1.

Works on assignment-based SDVRPs are rather "method-centric", i.e., they focus primarily on analyzing the power and versatility of RL methodology. Typical examples include ride hailing problems (see the example in Section 2.1 and Kullman et al. (2020), Wen et al. (2017), Wang et al. (2018), Oda and Joe-Wong (2018), Zhou et al. (2019), Singh et al. (2019), Tang et al. (2019), Al-Abbasi et al. (2019), Qin et al. (2020), Jintao et al. (2020), Liu et al. (2020), Mao et al. (2020) and Oda (2021)) and pure dispatching problems (see Chen et al. (2019), Kavuk et al. (2021) and Ding et al. (2021)). In both problems, vehicles are assigned to customer requests or repositioned on a grid. No tentative routes must be planned and feasibility of actions can be verified directly. For these types of problems, action and state spaces are similar to what is described in Fig. 3. The low problem complexity allows researchers to use the existing RL methodology to its full potential and incorporate complete state- and action-space information into their machinery. Several works learn state–action values in high-dimensional state spaces with the help of well-chosen network architectures and update procedures (Kullman et al., 2020; Chen et al., 2019; Jahanshahi et al., 2021). An example for efficient usage of state information can be found in the RL implementation of Kullman et al. (2020) for an electric ride hailing problem. They construct their network architecture to process the full state space as opposed to reducing the state space in conformation to a given approximation architecture (see linear VFAs or lookup tables). This becomes even more striking in the work of Li et al. (2021) that considers a dynamic pickup and delivery problem in an industrial setting. They do not resort to discretization of the service area and use a graph-based formulation instead. Based on the service area graph, they construct a graph neural network as VFA, effectively integrating the service area's characteristics in their network architecture. In summary, works on assignment-based SDVRPs are already taking advantage of the available RL methodology. For that reason, we focus on highlighting the potential of RL for route-based SDVRPs.

Works on route-based SDVRPs are rather "model-centric" as they aim to model as many real-world aspects as possible. For these problems, the direct application of RL methodology is often impeded by two major obstacles. First, each state entails the currently planned routes which results in the state information being high-dimensional, of dynamic size (the number of routes and stops in the routes varies), and of sequential nature. All in all, this makes it challenging to represent state information in a format suitable for most approximation architectures, e.g., a real-valued vector. Second, the action space is large and combinatorial, i.e., not enumerable in reasonable time, and often so complex that even finding a feasible routing action is challenging. Supplementary actions regarding customer acceptance, partial fulfillment of demand, choice of transport modality, or relocation of vehicles may further complexify the action space. Most RL methods cannot deal with the large number of combinatorial actions in a state and it is difficult to assert that actions constructed by RL are feasible. To address these challenges, researchers have consistently resorted to two simplifications: *state–space aggregation* and *action-space restriction*. We will detail both in the following and give examples.

3.2.1. State–space aggregation

State–space aggregation addresses the challenge of the dynamic, combinatorial state space when applying RL methodology for SDVRP. The high dimensionality makes naive tabulation approaches of state–action values impossible and encourages underfitting when carelessly employing NNs as functional approximators. Thus, previous works on SDVRP heavily aggregated the state space based on expert knowledge to meaningful features. These features are often associated with route details such as arrival times or slack (see Ulmer et al. (2018a)) and enable the approximation architecture to learn the value of state–action pairs on the aggregated state space (see Agussurja et al. (2019)). The level of detail of the employed state–space aggregation is often linked directly to the complexity of the approximation architecture. Lookup tables typically are limited to two to three features (see Schmid (2012), Ulmer et al. (2018a,b), Brinkmann et al. (2019), Ulmer et al. (2019), Soeffker et al. (2019) and Basso et al. (2022)), linear approximators and hand-crafted VFAs (see Maxwell et al. (2010), Meisel et al. (2011), van Heeswijk et al. (2019) and Al-Kanj et al. (2020)) are able to incorporate finer state–action detail, but require a-priori knowledge of the nature of the value function, and deep neural network architectures allow for the effective usage of more features which is akin to a finer discretization (see Chen et al. (2022, 2020) and Ma et al. (2021)), but training them is computationally expensive and requires a large number of observations. Essentially, when solving SDVRPs with routing actions, conventional approximation architectures require that we perform a state–space aggregation and transform the state and action information into a format suitable for our VFA; typically, a real-valued vector that is of static size over all states and actions. Hence, we can define a state space aggregation as a projection from the state–action space to a real-valued vector space $\mathcal{A} : \mathcal{S} \times \mathcal{X} \rightarrow \mathbb{R}^l$. Using the state–space aggregation $\mathcal{A}$, we can express our value function approximation as $V(S_k, x_k) \approx \hat{V}(\mathcal{A}(S_k, x_k))$. Ideally, the state–space aggregation is quite comprehensive to enable efficient learning, but still allows for an accurate approximation and clear differentiation of states with different values.

In the following, we illustrate the concept of state–space aggregation for the ride sharing example. Here, the state S_k entails the current time t_k, the set of all planned routes $\bar{\Omega}_k$, the unserved requests given by their pickup locations P_k, dropoff locations D_k, number of passengers $q_k \in \mathbb{N}_k^N$ and deadlines δ_{ik} for $i \in D_k$, and the travel time matrix tt. An action x_k for state S_k is given by a set of routes $x_k \subset \Omega(S_k)$. In

particular, the set of planned routes $\bar{\Omega}_k$ is of dynamic size and entails sequential data that is difficult to embed in a vector space. A sensible state–space aggregation $\mathcal{A}$ could include temporal information, fleet information, and spatial information. Temporal information could be represented by the current time t_k as this is a good indicator of the number of future requests and reward opportunities. That is, the further time is in the horizon, the fewer opportunities there are for future requests. Fleet information could be given by summary statistics on the route length of each vehicle in the fleet given an action $x_k \subset \Omega(S_k)$. Longer routes indicate a higher current workload that will likely lead to less reward in the future. The route length B^v of each vehicle $v \in \mathcal{V}$ can be defined as the vehicle's arrival at the last stop

$$B^v = \max_{j \in N} \sum_{i \in D_k} x^v_{ijk}(B^v_i + \mathrm{tt}_{ij}). \tag{22}$$

Because the number of vehicles can be large, instead of individual consideration, we need to use summarizing statistics that aggregate the vehicles. The corresponding summary statistics could include the minimum route length

$$B^- = \min_{v \in \mathcal{V}} B^v,$$

the maximum route length

$$B^+ = \max_{v \in \mathcal{V}} B^v,$$

and the mean route length

$$\bar{B} = \frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} B^v.$$

Another important aspect might be the spatial distribution of the vehicles in the service area indicating how well the area and future demand can be covered. Thus, spatial information could be represented by the coverage of the fleet given by an estimate of the maximum response time to a fixed set of dummy requests $\bar{P}$ in the service area

$$\mathrm{RT} = \max_{i \in \bar{P}} \min_{v \in \mathcal{V}} B^v_{j(v)} + \mathrm{tt}_{j(v)i},$$

where $j(v) \in N$ denotes the last planned stop in the route of vehicle $v \in \mathcal{V}$. In summary, for our example, we obtain the following state–space aggregation:

$$\mathcal{A}(S_k, x_k) = (t_k, B^-, B^+, \bar{B}, RT)^\intercal$$

and we evaluate a state–action-tuple by

$$V(S_k, x_k) \approx \hat{V}(t_k, B^-, B^+, \bar{B}, \mathrm{RT}|\theta).$$

For this example, $\hat{V}(\cdot|\theta)$ could take a linear form (as suggested, e.g., in Rivera and Mes (2017)) with trainable parameters θ

$$\hat{V}(t_k, B^-, B^+, \bar{B}, \mathrm{RT}|\theta) = \theta_0 + \theta_1 \cdot t_k + \theta_2 \cdot B^- + \theta_3 \cdot B^+ + \theta_4 \cdot \bar{B} + \theta_5 \cdot \mathrm{RT}.$$

Importantly, the size of this state–space aggregation is independent of the number of vehicles or customer requests. Therefore, a value function approximation based on this state–space aggregation may be transferred to instances with deviating number of vehicles and demand.

3.2.2. Action-space restriction

Action-space restriction addresses the challenge of the complex, combinatorial action space when applying RL for SDVRPs. Some action-space restrictions are described as a mapping $\mathcal{R} : \mathcal{X} \to \mathcal{X}$ that reduces the action space $\mathcal{X}$ to a subset $\mathcal{R}(\mathcal{X}) \subset \mathcal{X}$ of enumerable candidate actions. Other action space restrictions are better expressed as a factorization of the action space $\mathcal{X} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_n$ (e.g., Ulmer et al. (2018a), Chen et al. (2020) and Lei et al. (2020)). This factorization is then solved sequentially as follows. First, search the action space $\mathcal{X}_1$ to obtain partial solution $x^{(1)}_k$. Second, construct the next action space as $\{x^{(1)}_k\} \times \mathcal{X}_2$ to obtain $x^{(2)}_k$. We proceed in this fashion until we obtain all partial actions. Finally, we construct x_k from the partial actions $x^{(1)}_k, \ldots, x^{(n)}_k$. We summarize the procedure in Algorithm 2.

Input: Factorization $\mathcal{X} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_n$ and corresponding solution methods $\mathrm{SOL}_1, \ldots, \mathrm{SOL}_n$
Output: x_k
Set $\mathcal{X}' = \mathcal{X}_1$
for $(i = 1, \ldots, n)$ **do**
  $x^{(i)}_k = \mathrm{SOL}_i(\mathcal{X}')$
  $\mathcal{X}' \leftarrow \mathcal{X}' \times \{x^{(i)}_k\}$
end
$x_k \leftarrow (x^{(1)}_k, \ldots, x^{(n)}_k)$

Algorithm 2: Sequential action-space restriction.

In other words, such an action-space restriction separates a complex action space into a sequence of simpler tasks and solves them sequentially. Some tasks are solved by RL, others are solved by established heuristics. Ideally, the factorization is done in a way that the part evaluated by RL has a major impact on the future value, but is still easily enumerable.

For example, our ride sharing problem in Section 2.1 requires simultaneous acceptance, assignment, and routing actions. Acceptance and assignments decisions might be particularly important as they influence immediate reward and overall fleet flexibility (Ulmer et al., 2018a). An action-space restriction could factorize this combined action space into two sequential action spaces: first an assignment action (including the rejection of requests) and, second, a routing action. The simpler, enumerable acceptance and assignment component might be attended to by an RL algorithm, while the complex routing actions, are address by established VRP heuristics. For example, a sensible action-space restriction for the ride sharing problem is given by iteratively assigning and integrating new requests to existing routes. A possible procedure is given in Algorithm 3.

Input: new requests $R^{\mathrm{new}} := \{(i, i+n, \delta_{ik}) \mid i \in P^{\mathrm{new}}_k := P_k \setminus P_{k-1}\}$, planned routes $\bar{\Omega}_k$, insertion routing policy INSERT
Output: x_k
$x_k \leftarrow \bar{\Omega}_k$
for $(i, i+n, \delta_{ik}) \in R^{new}$ **do**
  $x_k \leftarrow \mathrm{INSERT}((i, i+n, \delta_i), x_k)$
end

Algorithm 3: Action-space restriction for an SDPDP.

Algorithm 3 proceeds as follows. We initialize our action x_k to the set of currently planned routes $\bar{\Omega}_k$. We then iterate over all new requests $(i, i+n, \delta_{ik}), i \in P^{\mathrm{new}}_k$. In every step, we insert the pickup node i and our delivery node $i+n$ into one route of our current set of routes x_k (asserting feasibility) and update x_k accordingly. We return x_k once we finished iterating over all new requests.

An example in the literature of such an action-space restriction is (Chen et al., 2022), who face a same-day delivery problem with a heterogeneous fleet of drones and vehicles. A deep Q-learning policy learns whether to serve a delivery request by drone, vehicle, or reject it altogether. The assignment onto the fleet of vehicles or drones as well as the integration of deliveries into existing routes are performed by hand-crafted heuristics. In another example, Joe and Lau (2020) learn routing actions for a dynamic delivery problem. They assume that requests are already assigned to the fleet at the start of the day. Their policy combines Q-learning and simulated annealing to re-route vehicles when customers cancel or when new customers are assigned to a vehicle.

Multi-agent RL frameworks constitute another form of action-space restriction. In such frameworks, each vehicle in the fleet is modeled as

Table 1
Classification of literature on RL for SDVRP.

	Literature	State-space aggregation	Action-space restriction	VFA	PG
Assignment-based SDVRPs	Maxwell et al. (2010)	✓	✓	✓	
	Schmid (2012)	✓	✓	✓	
	Wen et al. (2017)		✓	✓	
	Wang et al. (2018)		✓	✓	
	Nazari et al. (2018)				✓
	Kool et al. (2019)				✓
	Oda and Joe-Wong (2018)			✓	
	Chen et al. (2019)				✓
	Singh et al. (2019)			✓	
	Al-Abbasi et al. (2019)			✓	
	Balaji et al. (2019)			✓	
	Tang et al. (2019)			✓	
	Zhou et al. (2019)			✓	
	Brinkmann et al. (2019)	✓		✓	
	van Heeswijk et al. (2019)	✓	✓	✓	
	Mao et al. (2020)				✓
	Liu et al. (2020)			✓	
	Kullman et al. (2020)			✓	
	Qin et al. (2020)			✓	
	Jintao et al. (2020)			✓	✓
	Al-Kanj et al. (2020)		✓	✓	
	Jahanshahi et al. (2021)			✓	
	Kavuk et al. (2021)			✓	
	Li et al. (2021)			✓	
	Oda (2021)			✓	
	Ding et al. (2021)			✓	
	Beirigo et al. (2022)			✓	✓
Route-based SDVRPs	Ulmer et al. (2018a)	✓	✓	✓	
	Ulmer et al. (2018b)	✓	✓	✓	
	Agussurja et al. (2019)	✓	✓	✓	
	Soeffker et al. (2019)	✓	✓	✓	
	Ulmer et al. (2019)	✓	✓	✓	
	Biggs and Perakis (2020)	✓		✓	
	Ulmer and Thomas (2020)	✓	✓	✓	
	Ulmer (2020)	✓	✓	✓	
	Lei et al. (2020)	✓	✓	✓	
	Chen et al. (2020)		✓	✓	
	Chen et al. (2022)		✓	✓	
	Ma et al. (2021)		✓	✓	✓
	Basso et al. (2022)	✓	✓	✓	
	Zhang et al. (2022)	✓		✓	

an independent agent. Thus, decisions are not derived for the entire fleet, but on an agent-level and the action space is decomposed accordingly. Cooperation of agents in the fleet is only indirectly incorporated. For example, Chen et al. (2019) tackle a dynamic courier dispatching problem by modeling vehicles as independent agents embedded in multi-agent framework. Each agent's decision policy is given by an actor–critic policy. While they use purely de-centralized decision making, they incentivize cooperative behavior by reshaping the reward function.

3.3. Summary

We summarize works on RL for SDVRP in Table 1. The table is divided into two parts that correspond to assignment-based SDVRPs (e.g., the ride hailing problem) and route-based SDVRPs (e.g. the ride sharing problem). Table 1 is structured as follows: The column "Literature" cites the work. The column "State–space aggregation" indicates whether the work aggregates the state information. The column "Action-space restriction" denotes whether an action-space restriction was performed instead of searching or enumerating the entire action space. The column "VFA" denotes if the work uses a VFA-based RL method and the column "PG" indicates whether the work employs a policy gradient method.

The table confirms our previous assessment. Work on assignment-based SDVRPs already draws from a wide spectrum of RL methods, using both value function approximation and policy gradient methods without restricting the action space or aggregating the state space. In contrast, work on route-based SDVRPs rely heavily on state–space aggregation and action-space restriction to accommodate VFA-based RL solution methods. Based on our findings, we see a major opportunity in solution methods that employ RL methods for route-based SDVRPs without fully depending on simplifications such as state–space aggregation or action-space restriction. We further see that PG-methods are still very underutilized in route-based SDVRP research.

4. Opportunities for future work

Route-based SDVRPs require powerful tools to search the combinatorial, complex routing action spaces and evaluate the value of state–action pairs based on high-dimensional state information. As we highlighted in the previous section, there is no work in the literature that simultaneously searches and evaluates the full action space of route-based SDVRPs. Furthermore, the potential of policy gradient methods has barely been explored for SDVRPs. For that reason, we suggest two approaches for solving route-based SDVRPs that fill this gap and that we see as particularly promising. In both approaches, we model each decision point as a static VRP to tap into the rich pool of VRP solvers. We do not ignore the stochasticity and dynamism of the SDVRP as the static VRP formulation is modified in each state to account for future states. In the first approach, anticipation is introduced into the static formulation by extending the objective function of the MILP by a complex (piecewise-)linear VFA that estimates the future cumulative reward given the current state S_k and action x_k. In the second approach, anticipation is introduced by modifying the

constraints and objective of the MILP to reserve current resources for future states or adjust the reward vector to better reflect the dynamic nature of the problem. This modification is implemented directly by a policy gradient method. In the following, we explain both approaches and give directions on how to apply them for the example of ride sharing as described in Section 2.1.

4.1. VFA extension of the static objective

In our first approach, we avoid an action-space restriction, but still require a state–space aggregation. Extending the objective function of the MILP by a VFA boils down to two steps. In the first step, we construct the MILP given by Problem (1) in every decision point $k \in \{0, \dots, K\}$ of the MDP and extend its objective function by a VFA term:

$$\begin{aligned} \text{minimize} \quad & \sum_{r \in \Omega(S_k)} c_{rk} \cdot x_{rk} + \hat{V}\big(\mathcal{A}(S_k, x_k)\big) \\ \text{subject to} \quad & \sum_{r \in \Omega(S_k)} a_{irk} x_{rk} = 1, && \forall i \in N \\ & x_{rk} \in \{0,1\}, && \forall r \in \Omega(S_k) \end{aligned} \tag{23}$$

In order to ensure that our extension of Problem (1) remains linear, we assume that our value function approximation $\hat{V} : \mathbb{R}^l \to \mathbb{R}$ as well as the state–space aggregation $\mathcal{A} : S \times \mathcal{X} \to \mathbb{R}^l$ are piecewise-linear. In the second step, we take advantage of powerful VRP solvers to efficiently search the action space as defined in Problem (23) and evaluate each action based on its immediate reward $c_k^\intercal x_k$ and the expected future reward $\hat{V}\big(\mathcal{A}(S_k, x_k)\big)$.

There is work that employs such an approach for related problems. Chen et al. (2018), Topaloglu and Powell (2006), Lei et al. (2020), Beirigo et al. (2022) and Biggs and Perakis (2020) combine linear or (simple) piece-wise linear value function approximations (VFAs) with MILP solvers to search the action space in each decision point. However, linear VFAs are known to poorly approximate state–action values for routing problems (see Ulmer and Thomas (2020)). An alternative is combining MILP solvers with more complex piecewise-linear functions that are capable of approximating the value function well. Most feed-forward neural networks rely on ReLU non-linearities which are, in fact, piece-wise linear as the ReLU function is given by $\text{ReLU}(x) := \max(0, x)$. All other components of the neural network can be expressed as affine functions. Therefore, we may think of trained ReLU neural networks as piece-wise linear functions which we may plug into the MILP. In the past, the composition of affine functions and ReLU non-linearities where modeled as big-M constraints which led to inefficient MILPs. Recently, Anderson et al. (2020) and Tsay et al. (2021) derived strong MILP formulations for trained ReLU neural networks. Delarue et al. (2020) use their formulations and guide MILP solvers on static capacitated vehicle routing problems.

We now illustrate this concept with our example of ride sharing. As state–space aggregation we could use the state–space aggregation $\mathcal{A}$ that we defined in Section 3.2.1 as it is piece-wise linear. By using the variables defined in Constraints (7) and (9), we could model the state–space aggregation as

$$B^v = \max_{j \in N} \sum_{i \in D_k} x^v_{ijk}(B^v_i + \text{tt}_{ij}) \qquad \forall v \in \mathcal{V} \tag{24}$$

$$B^- = \min_{v \in \mathcal{V}} B^v \tag{25}$$

$$B^+ = \max_{v \in \mathcal{V}} B^v \tag{26}$$

$$\bar{B} = \frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} B^v \tag{27}$$

$$RT = \max_{i \in \bar{P}_k} \min_{v \in \mathcal{V}} B^v + \sum_{j \in D_k} t_{ji} \Big(\sum_{m \in N} x_{mjk} - \sum_{l \in N} x_{jlk} \Big) \tag{28}$$

Constraints (24) set the route length of each vehicle $v \in \mathcal{V}$. In Constraints (28), we express the travel time from the last stop $j(v)$ in the route of vehicle v to our dummy node i as

$$t_{j(v),i} = \sum_{j \in D_k} t_{ji} \Big(\sum_{m \in N} x^v_{mjk} - \sum_{l \in N} x^v_{jlk} \Big).$$

Now that we have defined our state–space aggregation, we may define our value function approximation and model it as a set of linear constraints. We could use a feed-forward neural network with rectified linear unit (ReLU) activations where $\text{ReLU}(x) := \max(x, 0)$. Such a neural network can be described by its s layers and corresponding number of neurons m_i, $i \in \{1, \dots, s\}$ in layer i. The neural network can be recursively defined by the output n^i of layer i. More specifically, we define the output of neuron j in layer i as

$$n^i_j := \text{ReLU}\big(w^{i,j} \cdot n^{i-1} + b^{i,j}\big). \tag{29}$$

Here, $\{w^{i,j}, b^{i,j} \mid i \in \{1, \dots, s\}, j \in \{1, \dots, m_i\}\}$ is the set of trainable parameters of the neural network. Each ReLU activation can be written as a big-M constraint.[2] Let M^+ and M^- denote the maximum and minimum value that our affine function $w^{i,j} \cdot x^{i-1} + b^{i,j}$ attains over its input domain. Then, we write the output of neuron j in layer i as

$$n^i_j \geq w^{i,j} \cdot n^{i-1} + b^{i,j} \tag{30}$$

$$n^i_j \leq w^{i,j} \cdot n^{i-1} + b^{i,j} - M^- \cdot (1 - z) \tag{31}$$

$$n^i_j \leq M^+ \cdot z \tag{32}$$

$$z \in \{0, 1\} \tag{33}$$

For the sake of example, we illustrate this for a network consisting of an input layer with a ReLU activation that maps our state space aggregation $\mathcal{A}(S_k, x_k) \in \mathbb{R}^5$ to a hidden state $n^1 \in \mathbb{R}^{10}$ and an output layer that maps n^1 to a scalar value $\hat{V}_k \in \mathbb{R}$ that approximates $V(S_k, x_k)$. We could model our value function approximation accordingly as

$$n^1_j \geq w^{1,j}(t_k, B^-, B^+, \bar{B}, RT)^\intercal + b^{1,j} \qquad \forall j \in 1, \dots, 10 \tag{34}$$

$$n^1_j \leq w^{1,j}(t_k, B^-, B^+, \bar{B}, RT)^\intercal + b^{1,j} - M^-(1 - z) \qquad \forall j \in 1, \dots, 10 \tag{35}$$

$$n^1_j \leq M^+ z \qquad \forall j \in 1, \dots, 10 \tag{36}$$

$$\hat{V}_k = w^{2,1} n^1 + b^{2,1} \tag{37}$$

$$z \in \{0, 1\} \quad . \tag{38}$$

Putting our formulation of the action space given by Constraints (4)–(12), our formulation of the state–space aggregation given by Constraints (24)–(28), and our formulation of the value function approximation given by Constraints (34)–(38) together, we yield the following optimization problem to be solved in every decision point

$$\text{maximize} \quad \sum_{v \in \mathcal{V}} \sum_{j \in \bar{P}_k} x^v_{ijk} + \hat{V}_k \tag{39}$$

$$\text{subject to} \quad \text{Constraints (4)–(12), (24)–(28), (34)–(38).} \tag{40}$$

The major advantage of this approach is that we are able to search the complex, combinatorial action space while evaluating the future impact of each action. However, this approach might come with three drawbacks. First, we require a piecewise-linear state–space aggregation which strongly limits the level of state details considered. Second, we systematically search for approximation errors in our value function approximation by including it in the objective function of a MILP. Third, training of the VFA might be slow when an extensive number of iterations is needed and when the network architecture is large. Thus, practicioners should put strong emphasis on creating small VFA architectures to reduce runtime while ensuring that the architecture's

[2] Note that a big-M formulation of a composition of a ReLU function and an affine function is not sharp and we only use it for ease of notation. For ideal formulations, we refer the reader to Anderson et al. (2020).

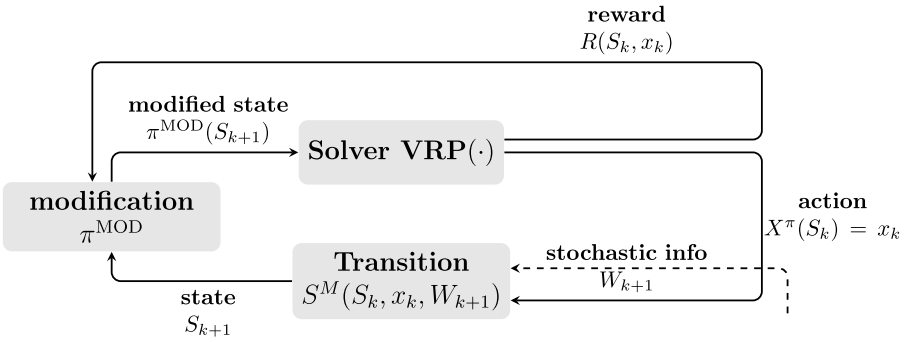

Fig. 5. Schematics of the modification approach.

output is "smooth" to avoid unexpected fluctuations in the objective function. In our second solution approach, we aim to avoid such concerns.

4.2. PG modification of objective and constraints

Our second approach does not rely on an state–space aggregation nor an action-space restriction. In this approach, we modify the constraints and objective of a MILP by a policy gradient method before solving it. The modification of constraints and objective is performed implicitly by mapping a given state S_k to an altered state $\pi^{\text{MOD}}(S_k)$ with the help of a modification policy π^{MOD} trained in policy gradient fashion. As such, the modification takes place in each decision point and is dependent on current state information. We summarize where and when the modification takes place in Fig. 5. The figure is structured analogously to Fig. 1, but comes with three adaptations. First, states S_{k+1} are no longer directly fed into the policy, but modified by the modification policy π^{MOD} in before (gray box, "modification"). Second, the policy is now given by a vehicle routing solver VRP$(\cdot)$ (gray box, "Solver"). Third, the rewards are no longer fed back to the policy (as this is a static solver now), but are provided to the modification policy, instead.

The state modification allows us to save resources for the future by strengthening constraints and by incentivizing or penalizing the acceptance of individual requests. This may be realized by decreasing the demand of individual requests, decreasing the deadline of individual requests, decreasing the capacity of individual vehicles, or changing the reward associated with individual requests. Modifications can be flexible and target various parts of the state information as long as $\Omega\big(\pi^{\text{MOD}}(S_k)\big) \subset \Omega(S_k)$ holds. In essence, this constitutes an extension of the CFA approaches presented in Section 2.2. The difference being that in existing CFA approaches modifications take place once on a strategical level, i.e., once for a given SDVRP instance. In our approach, we learn the modification on an operational level, i.e., individually for every state. Therefore, solving an SDVRP amounts to, first, modifying each state

$$S'_k := \pi^{\text{MOD}}(S_k),$$

second, constructing the resulting MILP

$$\begin{aligned} \text{minimize} \quad & \sum_{r \in \Omega(S'_k)} c_{rk} \cdot x_{rk} \\ \text{subject to} \quad & \sum_{r \in \Omega(S'_k)} a_{irk} x_{rk} = 1, \qquad \forall i \in N \\ & x_{rk} \in \{0,1\}, \qquad \forall r \in \Omega(S'_k), \end{aligned} \tag{41}$$

and, third, solving the MILP by a dedicated vehicle routing solver VRP : $\mathcal{S} \to \mathcal{X}$. Thus, the corresponding policy for solving the SDVRP is given by

$$\pi : \mathcal{S} \to \mathcal{X} \tag{42}$$

$$S_k \mapsto \text{VRP}\big(\pi^{\text{MOD}}(S_k)\big). \tag{43}$$

By modifying the state, we change not only the set of feasible routes, but also the rewards associated with routes. As such, a pair of routes $(r, r') \in \Omega(S_k) \times \Omega(\pi^{\text{MOD}}(S_k))$ with $r = r'$ might have different associated rewards $c_{rk} \neq c_{r'k}$.

We now present one possible MILP modification for the example of ride sharing. As described in Section 2.1, each state in the MDP of the ride sharing problem is given by the current time t_k, the set of tentative routes $\bar{\Omega}_k$, the unserved requests given by their pickup locations P_k, dropoff locations D_k, number of passengers $q_{ik} \in \mathbb{N}$ for $i \in P_k$, and deadlines δ_{ik} for $i \in D_k$, and the travel time matrix tt. The capacity of each vehicle is given by $Q^v \in \mathbb{N}$ for all $v \in \mathcal{V}$. The reward r_{ik} associated with each request is set to 1 if we accept the request and 0 else.

Appealing targets for a modification are the deadlines δ_{ik}, rewards r_{ik} associated with each request (in our ride sharing example we assumed that the reward per accepted request is 1), and the passenger capacity of each vehicle Q^v. Each of these modifications might shape the solution x_k in a state S_k in a unique way. Decreasing the deadline of specific requests in a state might prohibit consolidation of some requests or change the order in which requests are visited in a route. Increasing or decreasing rewards of individual requests might alter the acceptance of requests. Finally, decreasing the passenger capacity of vehicles might reserve passenger capacity for future, currently unknown, requests. As such, our modification policy π^{MOD} might take the form of

$$\pi : \begin{matrix} (t_k, \bar{\Omega}_k, P_k, D_k, q_k, tt)^\intercal \\ (\delta_k, r_k, Q)^\intercal \end{matrix} \mapsto \begin{matrix} (t_k, \bar{\Omega}_k, P_k, D_k, q_k, tt)^\intercal \\ (\delta_k, r_k, Q)^\intercal \odot x_k^{\text{MOD}}, \end{matrix} \tag{44}$$

where the state information $(t_k, \bar{\Omega}_k, P_k, D_k, q_k, tt)^\intercal$ remain unaltered and the state information $(\delta_k, r_k, Q)^\intercal$ are mapped to their element-wise product with a modification action $x_k^{\text{MOD}} \in \mathbb{R}^{2|D_k|+|\mathcal{V}|}$. The modification action x_k^{MOD} is the output of a neural network $\text{NN}^{\text{MOD}}(\cdot|\theta)$ parameterized by its weights θ. Specifically, we design the neural network such that its image is restricted to $[0,1]^{|D_k|} \times [-\infty, \infty]^{|D_k|} \times [0,1]^{|\mathcal{V}|}$. This ensures that deadlines can only be decreased, but remain positive, rewards can be arbitrarily increased and decreased and even mapped to a negative value (which directly implies that the request is not accepted), and capacities might be decreased but may not become negative. For practical application, we must further restrict the image to assert that all previously accepted customers can still be feasibly served. In the following, we write our modification policy as $\pi^{\text{MOD}}(\cdot|\theta)$ to emphasize that it is implicitly defined by our neural network $\text{NN}^{\text{MOD}}(\cdot|\theta)$. The parameters θ of NN^{MOD} may be trained by an adaption of the REINFORCE Algorithm 1. One possible procedure is given in Algorithm 4. The approach is analogous to the REINFORCE algorithm except that we also save the modification actions and update θ using the likelihood of the modification action x_k^{MOD} instead of the actual action x_k. The advantage of this approach is that we effectively avoid a state–space aggregation and an action-space restriction. This enables us to search the action space efficiently using full state information. On the other hand, we do not evaluate the future value of actions directly, but weave

anticipation of future states and actions into the constraints and reward vector of the MILP. In contrast to the previous VFA extension of the MILP, this does not complexify the MILP and may even simplify it, e.g., when modifying the reward associated with a request to a negative value we effectively exclude this request from the problem.

Input: A modification policy π^{MOD} which is based on a neural network $\text{NN}^{\text{MOD}}(\cdot, \theta)$ with initial weights θ, a vehicle routing solver $\text{VRP}(\cdot)$, a stepsize $\alpha > 0$, and a discount factor $0 < \gamma \leq 1$.
Output: Trained policy parameters θ.
while *True* **do**
  Generate an episode $S_0, x_0^{\text{MOD}}, x_0, R(S_0, x_0), \dots, S_K, x_K^{\text{MOD}}, x_K, R(S_K, x_K)$, following $\text{VRP}\left(\pi^{\text{MOD}}(\cdot|\theta)\right)$
  for $k = 0, \dots, K-1$ **do**
    $v_k = \sum_{t=k+1} \gamma^{t-k-1} R(S_t, x_t)$
    $\theta \leftarrow \theta + \alpha \gamma^k v_k \nabla \ln \text{NN}^{\text{MOD}}(x_k^{\text{MOD}}|S_k, \theta)$
  end
end

Algorithm 4: Training the modification policy.

5. Closing remarks

Instant delivery, ride sharing, and restaurant meal delivery are booming and stochastic dynamic vehicle routing is a growing field because of it. The increasing dynamism and stochasticity of these services pose unique challenges that have only been partially addressed in the past. We believe that reinforcement learning is a particularly promising solution method to overcome these challenges as it yields fast, complex, and anticipatory decision policies. However, many reinforcement learning techniques, such as policy gradient methods, are untouched for route-based SDVRPs. A direct application of these techniques to route-based SDVRPs is rarely possible as they struggle to deal with the complex, combinatorial action space. Searching the action space is best performed by the large pool of effective VRP solvers. As such, there is untapped potential in combining reinforcement learning with established VRP solvers to solve route-based SDVRPs in an anticipatory and holistic manner. Thus, we strongly believe that future collaborations of computer science and operations research will bring forward the next generation of solution methods for route-based SDVRPs.

Data availability

No data was used for the research described in the article.

Acknowledgments

Florentin Hildebrandt's research is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), project 413322447. Marlin Ulmer's work is funded by the DFG Emmy Noether Programme, Germany, project 444657906. We gratefully acknowledge their support.

References

Agatz, N., Erera, A., Savelsbergh, M., Wang, X., 2012. Optimization for dynamic ride-sharing: A review. European J. Oper. Res. 223 (2), 295–303.
Agussurja, L., Cheng, S.-F., Lau, H.C., 2019. A state aggregation approach for stochastic multiperiod last-mile ride-sharing problems. Transp. Sci. 53 (1), 148–166.
Al-Abbasi, A.O., Ghosh, A., Aggarwal, V., 2019. Deeppool: Distributed model-free algorithm for ride-sharing using deep reinforcement learning. IEEE Trans. Intell. Transp. Syst. 20 (12), 4714–4727.
Al-Kanj, L., Nascimento, J., Powell, W.B., 2020. Approximate dynamic programming for planning a ride-hailing system using autonomous fleets of electric vehicles. European J. Oper. Res. 284 (3), 1088–1106.
Anderson, R., Huchette, J., Ma, W., Tjandraatmadja, C., Vielma, J.P., 2020. Strong mixed-integer programming formulations for trained neural networks. Math. Program. 1–37.
Archetti, C., Bertazzi, L., 2021. Recent challenges in routing and inventory routing: E-commerce and last-mile delivery. Networks 77 (2), 255–268.
Arslan, A.M., Agatz, N., Kroon, L., Zuidwijk, R., 2019. Crowdsourced delivery—A dynamic pickup and delivery problem with ad hoc drivers. Transp. Sci. 53 (1), 222–235.
Ausseil, R., Pazour, J.A., Ulmer, M.W., 2022. Supplier menus for dynamic matching in peer-to-peer transportation platforms. Transp. Sci. 56 (5), 1304–1326.
Balaji, B., Bell-Masterson, J., Bilgin, E., Damianou, A., Garcia, P.M., Jain, A., Luo, R., Maggiar, A., Narayanaswamy, B., Ye, C., 2019. Orl: Reinforcement learning benchmarks for online stochastic optimization problems. arXiv preprint arXiv:1911.10641.
Baldacci, R., Mingozzi, A., Roberti, R., 2011. New route relaxation and pricing strategies for the vehicle routing problem. Oper. Res. 59 (5), 1269–1283.
Basso, R., Kulcsár, B., Sanchez-Diaz, I., Qu, X., 2022. Dynamic stochastic electric vehicle routing with safe reinforcement learning. Transp. Res. Part E 157, 102496.
Beirigo, B.A., Schulte, F., Negenborn, R.R., 2022. A learning-based optimization approach for autonomous ridesharing platforms with service-level contracts and on-demand hiring of idle vehicles. Transp. Sci. 56 (3), 677–703.
Bent, R.W., Van Hentenryck, P., 2004. Scenario-based planning for partially dynamic vehicle routing with stochastic customers. Oper. Res. 52 (6), 977–987.
Biggs, M., Perakis, G., 2020. Dynamic routing with tree based value function approximations. Available at SSRN 3680162.
Bixby, R.E., 2012. A brief history of linear and mixed-integer programming computation. Doc. Math. (2012), 107–121.
Braekers, K., Ramaekers, K., Van Nieuwenhuyse, I., 2016. The vehicle routing problem: State of the art classification and review. Comput. Ind. Eng. 99, 300–313.
Branchini, R.M., Armentano, V.A., Løkketangen, A., 2009. Adaptive granular local search heuristic for a dynamic vehicle routing problem. Comput. Oper. Res. 36 (11), 2955–2968.
Brinkmann, J., Ulmer, M.W., Mattfeld, D.C., 2019. Dynamic lookahead policies for stochastic-dynamic inventory routing in bike sharing systems. Comput. Oper. Res. 106, 260–279.
Chen, X., Hewitt, M., Thomas, B.W., 2018. An approximate dynamic programming method for the multi-period technician scheduling problem with experience-based service times and stochastic customers. Int. J. Prod. Econ. 196, 122–134.
Chen, Y., Qian, Y., Yao, Y., Wu, Z., Li, R., Zhou, Y., Hu, H., Xu, Y., 2019. Can sophisticated dispatching strategy acquired by reinforcement learning?-a case study in dynamic courier dispatching system. arXiv preprint arXiv:1903.02716.
Chen, X., Ulmer, M.W., Thomas, B.W., 2022. Deep Q-learning for same-day delivery with vehicles and drones. European J. Oper. Res. 298 (3), 939–952.
Chen, X., Wang, T., Thomas, B.W., Ulmer, M.W., 2020. Same-day delivery with fairness. arXiv preprint arXiv:2007.09541.
Delarue, A., Anderson, R., Tjandraatmadja, C., 2020. Reinforcement learning with combinatorial actions: An application to vehicle routing. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (Eds.), Advances in Neural Information Processing Systems, Vol. 33. Curran Associates, Inc., pp. 609–620.
Desaulniers, G., Lessard, F., Hadjar, A., 2008. Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. Transp. Sci. 42 (3), 387–404.
Ding, Y., Guo, B., Zheng, L., Lu, M., Zhang, D., Wang, S., Son, S.H., He, T., 2021. A city-wide crowdsourcing delivery system with reinforcement learning. Proc. ACM Interact. Mobile, Wearable Ubiquit. Technol. 5 (3), 1–22.
Gendreau, M., Guertin, F., Potvin, J.-Y., Taillard, É., 1999. Parallel tabu search for real-time vehicle routing and dispatching. Transp. Sci. 33 (4), 381–390.
George, A., Powell, W.B., Kualkarni, S.R., 2008. Value function approximation using multiple aggregation for multiattribute resource management. J. Mach. Learn. Res. 9 (10).
Ghiani, G., Manni, E., Quaranta, A., Triki, C., 2009. Anticipatory algorithms for same-day courier dispatching. Transp. Res. Part E 45 (1), 96–106.
Goodson, J.C., Thomas, B.W., Ohlmann, J.W., 2017. A rollout algorithm framework for heuristic solutions to finite-horizon stochastic dynamic programs. European J. Oper. Res. 258 (1), 216–229.
Ichoua, S., Gendreau, M., Potvin, J.-Y., 2000. Diversion issues in real-time vehicle dispatching. Transp. Sci. 34 (4), 426–438.
Jahanshahi, H., Bozanta, A., Cevik, M., Kavuk, E.M., Tosun, A., Sonuc, S.B., Kosucu, B., Başar, A., 2021. A deep reinforcement learning approach for the meal delivery problem. arXiv preprint arXiv:2104.12000.
Jintao, K., Yang, H., Ye, J., et al., 2020. Learning to delay in ride-sourcing systems: a multi-agent deep reinforcement learning framework. IEEE Trans. Knowl. Data Eng..
Joe, W., Lau, H.C., 2020. Deep reinforcement learning approach to solve dynamic vehicle routing problem with stochastic customers. In: Proceedings of the International Conference on Automated Planning and Scheduling, Vol. 30. pp. 394–402.
Kavuk, E.M., Tosun, A., Cevik, M., Bozanta, A., Sonuç, S.B., Tutuncu, M., Kosucu, B., Basar, A., 2021. Order dispatching for an ultra-fast delivery service via deep reinforcement learning. Appl. Intell. 1–26.

Klapp, M.A., Erera, A.L., Toriello, A., 2018. The one-dimensional dynamic dispatch waves problem. Transp. Sci. 52 (2), 402–415.
Kool, W., Van Hoof, H., Welling, M., 2019. Attention, learn to solve routing problems!. In: International Conference on Learning Representations.
Kullman, N., Cousineau, M., Goodson, J., Mendoza, J., 2020. Dynamic ridehailing with electric vehicles. Transp. Sci. 56 (3), 775–794.
Lei, C., Jiang, Z., Ouyang, Y., 2020. Path-based dynamic pricing for vehicle allocation in ridesharing systems with fully compliant drivers. Transp. Res. B 132, 60–75.
Li, X., Luo, W., Yuan, M., Wang, J., Lu, J., Wang, J., Lü, J., Zeng, J., 2021. Learning to optimize industry-scale dynamic pickup and delivery problems. In: 2021 IEEE 37th International Conference on Data Engineering (ICDE). IEEE, pp. 2511–2522.
Liu, Z., Li, J., Wu, K., 2020. Context-aware taxi dispatching at city-scale using deep reinforcement learning. IEEE Trans. Intell. Transp. Syst..
Lysgaard, J., Letchford, A.N., Eglese, R.W., 2004. A new branch-and-cut algorithm for the capacitated vehicle routing problem. Math. Program. 100 (2), 423–445.
Ma, Y., Hao, X., Jianye, H., Lu, J., Liu, X., Tong, X., Yuan, M., Li, Z., Tang, J., Meng, Z., 2021. A hierarchical reinforcement learning based optimization framework for large-scale dynamic pickup and delivery problems. In: Thirty-Fifth Conference on Neural Information Processing Systems.
Mao, C., Liu, Y., Shen, Z.-J.M., 2020. Dispatch of autonomous vehicles for taxi services: A deep reinforcement learning approach. Transp. Res. C 115, 102626.
Maxwell, M.S., Restrepo, M., Henderson, S.G., Topaloglu, H., 2010. Approximate dynamic programming for ambulance redeployment. INFORMS J. Comput. 22 (2), 266–281.
Meisel, S., Suppa, U., Mattfeld, D., 2011. Serving multiple urban areas with stochastic customer requests. In: Dynamics in Logistics. Springer, pp. 59–68.
Mitrović-Minić, S., Krishnamurti, R., Laporte, G., 2004. Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows. Transp. Res. B 38 (8), 669–685.
Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al., 2015. Human-level control through deep reinforcement learning. Nature 518 (7540), 529–533.
Nazari, M., Oroojlooy, A., Snyder, L.V., Takáč, M., 2018. Reinforcement learning for solving the vehicle routing problem. In: Advances in Neural Information Processing Systems. pp. 9860–9870.
Oda, T., 2021. Equilibrium inverse reinforcement learning for ride-hailing vehicle network. In: Proceedings of the Web Conference 2021. pp. 2281–2290.
Oda, T., Joe-Wong, C., 2018. MOVI: A model-free approach to dynamic fleet management. In: IEEE INFOCOM 2018-IEEE Conference on Computer Communications. IEEE, pp. 2708–2716.
Powell, W.B., 2019. A unified framework for stochastic optimization. European J. Oper. Res. 275 (3), 795–821.
Psaraftis, H.N., Wen, M., Kontovas, C.A., 2016. Dynamic vehicle routing problems: Three decades and counting. Networks 67 (1), 3–31.
Pureza, V., Laporte, G., 2008. Waiting and buffering strategies for the dynamic pickup and delivery problem with time windows. INFOR: Inform. Syst. Oper. Res. 46 (3), 165–175.
Qin, Z., Tang, X., Jiao, Y., Zhang, F., Xu, Z., Zhu, H., Ye, J., 2020. Ride-hailing order dispatching at DiDi via reinforcement learning. INFORMS J. Appl. Anal. 50 (5), 272–286.
Riley, C., Van Hentenryck, P., Yuan, E., 2020. Real-time dispatching of large-scale ride-sharing systems: Integrating optimization, machine learning, and model predictive control. arXiv preprint arXiv:2003.10942.
Rivera, A.E.P., Mes, M.R., 2017. Anticipatory freight selection in intermodal long-haul round-trips. Transp. Res. Part E 105, 176–194.
Sáez, D., Cortés, C.E., Núñez, A., 2008. Hybrid adaptive predictive control for the multi-vehicle dynamic pick-up and delivery problem based on genetic algorithms and fuzzy clustering. Comput. Oper. Res. 35 (11), 3412–3438.
Schilde, M., Doerner, K.F., Hartl, R.F., 2014. Integrating stochastic time-dependent travel speed in solution methods for the dynamic dial-a-ride problem. European J. Oper. Res. 238 (1), 18–30.
Schmid, V., 2012. Solving the dynamic ambulance relocation and dispatching problem using approximate dynamic programming. European J. Oper. Res. 219 (3), 611–621.
Secomandi, N., 2001. A rollout policy for the vehicle routing problem with stochastic demands. Oper. Res. 49 (5), 796–802.
Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al., 2016. Mastering the game of go with deep neural networks and tree search. Nature 529 (7587), 484–489.
Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al., 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. arXiv preprint arXiv:1712.01815.
Singh, A., Al-Abbasi, A., Aggarwal, V., 2019. A reinforcement learning based algorithm for multi-hop ride-sharing: Model-free approach. In: Neural Information Processing Systems (Neurips) Workshop.
Soeffker, N., Ulmer, M.W., Mattfeld, D.C., 2019. Adaptive state space partitioning for dynamic decision processes. Bus. Inform. Syst. Eng. 61 (3), 261–275.
Soeffker, N., Ulmer, M.W., Mattfeld, D.C., 2022. Stochastic dynamic vehicle routing in the light of prescriptive analytics: A review. European J. Oper. Res. 298 (3), 801–820.
Song, Y., Ulmer, M.W., Thomas, B.W., Wallace, S.W., 2020. Building trust in home services—stochastic team-orienteering with consistency constraints. Transp. Sci. 54 (3), 823–838.
Sutton, R.S., Barto, A.G., 2018. Reinforcement Learning: An Introduction, second ed. MIT Press.
Tang, X., Qin, Z., Zhang, F., Wang, Z., Xu, Z., Ma, Y., Zhu, H., Ye, J., 2019. A deep value-network based approach for multi-driver order dispatching. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 1780–1790.
Thomas, B.W., 2007. Waiting strategies for anticipating service requests from known customer locations. Transp. Sci. 41 (3), 319–331.
Topaloglu, H., Powell, W.B., 2006. Dynamic-programming approximations for stochastic time-staged integer multicommodity-flow problems. INFORMS J. Comput. 18 (1), 31–42.
Toth, P., Vigo, D., 2014. Vehicle Routing: Problems, Methods, and Applications. SIAM.
Tsay, C., Kronqvist, J., Thebelt, A., Misener, R., 2021. Partition-based formulations for mixed-integer optimization of trained ReLU neural networks. arXiv preprint arXiv:2102.04373.
Ulmer, M.W., 2020. Dynamic pricing and routing for same-day delivery. Transp. Sci. 54 (4), 1016–1033.
Ulmer, M.W., Goodson, J.C., Mattfeld, D.C., Hennig, M., 2019. Offline–online approximate dynamic programming for dynamic vehicle routing with stochastic requests. Transp. Sci. 53 (1), 185–202.
Ulmer, M.W., Goodson, J.C., Mattfeld, D.C., Thomas, B.W., 2020a. On modeling stochastic dynamic vehicle routing problems. EURO J. Transp. Logist. 9 (2), 100008.
Ulmer, M.W., Mattfeld, D.C., Köster, F., 2018a. Budgeting time for dynamic vehicle routing with stochastic customer requests. Transp. Sci. 52 (1), 20–37.
Ulmer, M., Nowak, M., Mattfeld, D., Kaminski, B., 2020b. Binary driver-customer familiarity in service routing. European J. Oper. Res. 286 (3), 477–493.
Ulmer, M., Savelsbergh, M., 2020. Workforce scheduling in the era of crowdsourced delivery. Transp. Sci. 54 (4), 1113–1133.
Ulmer, M.W., Soeffker, N., Mattfeld, D.C., 2018b. Value function approximation for dynamic multi-period vehicle routing. European J. Oper. Res. 269 (3), 883–899.
Ulmer, M.W., Thomas, B.W., 2019. Enough waiting for the cable guy—Estimating arrival times for service vehicle routing. Transp. Sci. 53 (3), 897–916.
Ulmer, M.W., Thomas, B.W., 2020. Meso-parametric value function approximation for dynamic customer acceptances in delivery routing. European J. Oper. Res. 285 (1), 183–195.
Ulmer, M.W., Thomas, B.W., Campbell, A.M., Woyak, N., 2021. The restaurant meal delivery problem: Dynamic pickup and delivery with deadlines and random ready times. Transp. Sci. 55 (1), 75–100.
van Heeswijk, W.J., Mes, M.R., Schutten, J.M., 2019. The delivery dispatching problem with time windows for urban consolidation centers. Transp. Sci. 53 (1), 203–221.
Voccia, S.A., Campbell, A.M., Thomas, B.W., 2019. The same-day delivery problem for online purchases. Transp. Sci. 53 (1), 167–184.
Wang, Z., Qin, Z., Tang, X., Ye, J., Zhu, H., 2018. Deep reinforcement learning with knowledge transfer for online rides order dispatching. In: 2018 IEEE International Conference on Data Mining (ICDM). IEEE, pp. 617–626.
Wen, J., Zhao, J., Jaillet, P., 2017. Rebalancing shared mobility-on-demand systems: A reinforcement learning approach. In: 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC). Ieee, pp. 220–225.
Williams, R.J., 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach. Learn. 8 (3), 229–256.
Yildiz, B., Savelsbergh, M., 2019. Provably high-quality solutions for the meal delivery routing problem. Transp. Sci. 53 (5), 1372–1388.
Zhang, J., Luo, K., Florio, A.M., Van Woensel, T., 2022. Solving large-scale dynamic vehicle routing problems with stochastic requests. European Journal of Operational Research (ISSN: 0377-2217) http://dx.doi.org/10.1016/j.ejor.2022.07.015.
Zhou, M., Jin, J., Zhang, W., Qin, Z., Jiao, Y., Wang, C., Wu, G., Yu, Y., Ye, J., 2019. Multi-agent reinforcement learning for order-dispatching via order-vehicle distribution matching. In: Proceedings of the 28th ACM International Conference on Information and Knowledge Management. pp. 2645–2653.