WORKING PAPER SERIES

Reinforcement Learning Variants for Stochastic Dynamic Combinatorial Optimization Problems in Transportation

Florentin D. Hildebrandt / Alexander Bode
Marlin W. Ulmer / Dirk C. Mattfeld

Working Paper No. 06/2023

OTTO VON GUERICKE
UNIVERSITÄT
MAGDEBURG

FACULTY OF ECONOMICS
AND MANAGEMENT

Impressum (§ 5 TMG)

Herausgeber:
Otto-von-Guericke-Universität Magdeburg
Fakultät für Wirtschaftswissenschaft
Der Dekan

Verantwortlich für diese Ausgabe:
Florentin D. Hildebrandt, Alexander Bode
Otto-von-Guericke-Universität Magdeburg
Fakultät für Wirtschaftswissenschaft
Postfach 4120
39016 Magdeburg
Germany

http://www.fww.ovgu.de/femm

Bezug über den Herausgeber
ISSN 1615-4274

Reinforcement Learning Variants for Stochastic Dynamic Combinatorial Optimization Problems in Transportation

Florentin D. Hildebrandt
Chair of Management Science, Otto-von-Guericke-Universität Magdeburg, florentin.hildebrandt@ovgu.de

Alexander Bode
Institute for Decision Support, Technische Universität Braunschweig, alexander.bode@tu-braunschweig.de

Marlin W. Ulmer
Chair of Management Science, Otto-von-Guericke-Universität Magdeburg, marlin.ulmer@ovgu.de

Dirk C. Mattfeld
Institute for Decision Support, Technische Universität Braunschweig, d.mattfeld@tu-braunschweig.de

With rising customer expectations and increasing computational potential, many transportation services face real-time decision making in stochastic and dynamic environments. They often need to find and adapt complex plans that are effective now but also flexible with respect to future developments. Mathematically, these three challenges of searching the large and complex decision space for effective and flexible decisions are reflected in the three parts of the famous Bellman Equation, namely the reward function (effective), the value function (flexible), and the decision space (search). In the transportation literature, reinforcement learning (RL) has shown potential to quickly evaluate the reward- and value function of the Bellman Equation for a limited number of decisions but struggles to search a complex, constrained decision space immanent in most transportation problems. The question of how to combine the thorough search of the complex decision space with RL-evaluation techniques is still open. We propose three RL-based solutions, each inspired by one component of the Bellman Equation, to search for and evaluate decisions in an integrated manner. The first and second method learn to dynamically manipulate the reward function and decision space to encourage effective and flexible and prohibit inflexible decisions, respectively. The third method models the Bellman Equation as a mixed-integer linear programming formulation in which the value function is given by a neural network approximator. We compare our proposed solution methods in a structured analysis for carefully designed problem classes based on long-haul, medium-haul, and short-haul transportation logistics. We demonstrate the overall effectiveness of our methods compared to prominent benchmark methods and highlight how the methods' performances depend not only on the problem classes but also on the instances' parameterizations.

Key words: stochastic dynamic transportation problems, sequential decision processes, mixed integer linear programming, reinforcement learning, approximate dynamic programming

1. Introduction

Enabled by real-time information streams and catalyzed by increasing service expectations, transportation companies turn towards dynamic and real-time decision processes (Soeffker, Ulmer, and Mattfeld

2022). In such processes, decisions are made repeatedly in reaction to and in anticipation of future developments, e.g., new customers requesting service. The transportation applications are countless, ranging from long-haul trucking towards last-mile fulfillment. While the individual problems differ, they usually share the challenge that identifying a feasible decision in a state is a complex, often NP-hard, problem in itself. Typically decisions correspond to request selection, assignment, or routing and induce diverse constraints with respect to capacity, time, and customer requirements. OR-methods are required to search the space for effective decisions in a fast manner, given the need for real-time decision making. However, with the dynamism a new challenge arises: The value of a decision is unknown as it depends on future information and decisions. Decisions that are effective now may lead to inflexibility towards the future. This balance between immediate and future value is captured in the famous Bellman Equation (here shown for a maximization problem):

$$V(S_k) = \max_{x \in \mathcal{X}(S_k)} R(S_k, x) + V(S_k^x). \tag{1}$$

The value $V(S_k)$ of a state S_k is the maximum value that can be achieved over all decisions in this state $x \in \mathcal{X}(S_k)$. Thus, an optimal decision x maximizes the immediate reward $R(S_k, x)$ plus the expected future reward $V(S_k^x)$ conditioned on the decision taken. The complexity of the decision space is nicely hidden in the term $x \in \mathcal{X}(S_k)$. While the immediate reward function $R(S_k, x)$ is usually known, the value function $V(S_k^x)$ is highly uncertain and usually intractable. Since runtime in a state is limited, research increasingly focuses on learning the value function a priori by means of reinforcement learning (RL, also known under the terms approximate dynamic programming, value function approximation, approximate value iteration). Such RL-techniques use repeated offline simulations to approximate the value function which is then accessed for real-time decision making without additional runtime required. While this addresses the challenges of the second term of the Bellman Equation, the search of the complex decision space remains an issue. As a recent literature survey shows, most of the RL-work does not answer this challenge (Hildebrandt, Thomas, and Ulmer 2023). Often, the decision space is not fully searched but decomposed to "easy" and complex parts. The evaluation of the value function is then reduced to the easy parts while the complex parts are solved via heuristics.

In this research, we analyze how both the search of the decision space and its evaluation can be achieved via RL in an integrated manner. Based on the Bellman Equation, we identify three potential connection points for RL that still allow the search of the decision space with OR-methods: (1) adapting the decision space $\mathcal{X}(S_k)$, (2) adapting the reward function $R(S_k, x)$, (3) or directly learning and integrating the value function $V(S_k^x)$ into a mixed integer linear programming formulation of the Bellman Equation. Given the framework by Powell (2022), the first two belong to the class of cost function approximation (CFA) where reward function or decision space are manipulated to penalize or even prohibit inflexible decisions. While

the parameters of a CFA are usually determined by enumeration and are the same for all states, we propose using RL to search directly for state-specific parameters, either for the constraints of the decision space, or for the manipulation of the reward-function. For the third approach, approximating the value function $V(S_k^x)$, we use a neural network approximator that is integrated directly in the search of the decision space via a mixed integer linear programming formulation (MILP).

In theory, each of the three proposed concepts should be effective when solving dynamic problems in transportation with complex decision spaces. As there are at least hundreds of individual problems, we decide to specify and analyze our methodology for three representative problems: long-haul, medium-haul, and short-haul transportation with dynamic request selection. These problems reflect the common practice in transportation that a service provider serves dynamically occurring transportation requests with limited resources. Furthermore, the three problems allow for a careful and controlled increase in decision space complexity, from mere capacity constraints (all problems), over additional routing constraints (medium- and short-haul), to specific customer time-window constraints (short-haul). In a structured experimental analysis, we derive the following insights.

- All three methods show potential, but their performances depend on the problem and, equally important, the instance data characteristics.
- A direct approximation of the value function and an integrated optimization via the Bellman Equation provides the best results for the majority of test problems and instances but not for all. It should therefore be the first option when tackling dynamic problems in the field of transportation with RL.
- A decomposition of the complex decision space, as current practice in most of the RL-work in transportation, is inferior, sometimes to all three proposed methods.
- When deciding between a CFA and a decomposition of the decision space, the CFA is favorable for instances with high-dimensional decision spaces while the decision-space decomposition is preferable for instances with long decision horizons.

Our work makes the following contributions.

- We present a general classification of RL-approaches based on the Bellman Equation. This classification aids discussions of existing work but may also foster the development of future RL-methodology in our field.
- We detail an existing RL-approach and introduce two new RL-approaches to tackle the complex decision spaces immanent in problems from the area of transportation. All three approaches are general and, thus, applicable to a broader range of problems.
- We specify the three approaches for our three problems which building blocks are fundamental for many dynamic problems in transportation.
- We present analytical results on the expressiveness and convergence of our three methods.

- We assess the performance of our methods using established ADP methods from the literature as well as a perfect information lower bound.
- Our computational study does not only show that our RL-methods provide effective solutions for the individual problems, it also allows general insights in the interplay of methods, problem characteristics, and instance data.

The remainder of this work is structured as follows. Section 2 briefly summarizes the current state in optimization for dynamic problems in transportation. Section 3 motivates and defines the three fundamental problems we address in our computational study. Section 4 presents and analysis the three general RL-methods we propose. It also presents their specification for the three defined problems. Section 5 provides the details of our experimental study. Section 6 analyzes and discusses the policies' performance for the problems. Finally, Section 7 concludes our work and identifies future research opportunities.

2. Literature

Historically, solution methods for stochastic dynamic transportation problems fall either under the umbrella of static (re-)optimization or one of the policy classes of approximate dynamic programming (Soeffker, Ulmer, and Mattfeld 2022). Early works on combinatorial dynamic transportation problems treated decision states in the dynamic problem as static sub-problems on a rolling-horizon basis in order to exploit well-known mixed integer linear programming (MILP) methodologies (Gendreau et al. 1999, Ichoua, Gendreau, and Potvin 2000, Yang, Jaillet, and Mahmassani 2004, Branchini, Armentano, and Løkketangen 2009, Chen and Xu 2006). They focused on constructing "optimal" (for the current state of information only) decisions in a timely manner whenever information is updated in the decision process. Subsequent work additionally aimed to derive decisions that lead to better future values, e.g., decisions that induce flexible states (Benyahia and Potvin 1998, Mitrović-Minić and Laporte 2004, Thomas 2007, Pureza and Laporte 2008, Zehtabian, Larsen, and Wøhlk 2022). These methods are mostly characterized as hand-crafted (or rule-of-thumb) policies and fall under the domain of *policy function approximations* (PFAs). Follow-up work combined both concepts by integrating a PFA into static optimization procedures. Such *cost function approximations* (CFAs) manipulate the constraints or the objective function of the static optimization procedure in a non-state dependent manner (Riley, Van Hentenryck, and Yuan 2020, Ulmer et al. 2020, 2021). For example, Riley, Van Hentenryck, and Yuan (2020) add an artificial penalty for unserved requests for the ride-hailing problem while Ulmer et al. (2021) add a buffer to a deadline-constraint in a restaurant-meal-delivery problem to avoid delayed deliveries. Static optimization procedures, PFAs, and CFAs are all analytical in nature and do not evaluate decisions with regard to future uncertainty. In contrast, data-driven approaches incorporate estimations of stochasticity and future uncertainty into the decision policy. These approaches either sample stochastic information online or learn the value of decisions offline. Sampling-based approaches, such as *multiple-scenario approaches* (MSA) (Bent

and Van Hentenryck 2004) or *dynamic scenario hedging heuristics* (DSHH), construct an anticipatory solution by optimizing over a set of sampled future scenarios. Other sampling-based approaches, such as the *post-decision rollout algorithm* (RA) (Goodson, Thomas, and Ohlmann 2017) or *Monte Carlo tree search* (MCTS), thoroughly evaluate a small subset of candidate decisions by a forward simulation of their impact on the system (Secomandi 2001, Ulmer et al. 2019).

However, augmenting the decision problem by integrating sampled stochastic information before solving it online is often computationally too costly for large-scale applications. In contrast, *reinforcement learning* (RL) approaches learn the value of decisions in a given state in an extensive offline simulation to be able to instantly evaluate decisions when deployed online. The long-term value of decisions in a given state is learned either directly by means of *value function approximations* (VFA) or indirectly by means of *policy gradient* (PG) methods (Hildebrandt, Thomas, and Ulmer 2023). The focus of these RL methods is on the evaluation of decisions and they generally struggle to search vast and complex decision spaces. Therefore, sensible decision-space decompositions or a-priori-generated sets of candidate decisions are required to directly employ RL methods for stochastic dynamic transportation problems (Chen, Hewitt, and Thomas 2018, Ulmer, Mattfeld, and Köster 2018, Ulmer, Soeffker, and Mattfeld 2018, Lei, Jiang, and Ouyang 2019, Soeffker, Ulmer, and Mattfeld 2019, Ulmer et al. 2019, Agussurja, Cheng, and Lau 2019, Ulmer and Thomas 2020, Ulmer 2020, Ma et al. 2021, Chen, Ulmer, and Thomas 2022, Basso et al. 2022, Chen et al. 2023, Akkerman, Mes, and van Jaarsveld 2022).

In summary, the described methods provide a provenly solid groundwork to build on but each comes with its own significant shortcoming. Static re-optimization, PFA, and CFA methods do not explicitly evaluate decision with regard to future uncertainty, sampling-based approaches such as MSA, DSHH, RA, MCTS do not scale well due to their computational complexity, and off-the-shelf RL methods such as VFA and PG are not designed to search complex decision spaces. Motivated by the methods' shortcomings, we envision three paths leading from this groundwork towards *searching* a decision space while *evaluating* decisions' future impact. The paths correspond to the application of reinforcement learning to each one of the three elements of the Bellman Equation (1), namely the reward function $R(S, x)$, the decision space $\mathcal{X}(S_k)$, and the value function $V(S_k^x)$. The first and second approach train a policy gradient method to dynamically adjust the reward function or decision space based on current state information to induce efficient and flexible decisions. The third approach directly approximates the value function by a neural network which is integrated in a mixed integer linear program formulation representing the Bellman Equation to search the complex decision space. To our knowledge, there is no work that follows the first or second path, i.e., that learns to dynamically manipulate the reward function or decision space of the Bellman Equation. However, there is research related to the third path of approximating the value function and integrating the approximation into exact solvers. For the field of static optimization, Delarue,

Anderson, and Tjandraatmadja (2020), Papalexopoulos et al. (2022) integrate a piecewise affine neural network in a MÌLP model to heuristically solve combinatorial optimization problems including capacitated vehicle routing problems. For the field of stochastic dynamic optimization, there are works that employ linear approximations (Rivera and Mes 2017, Heinold, Meisel, and Ulmer 2022), tree-based approximations (Biggs and Perakis 2020), and even neural-network approximations (Silva, Pedroso, and Viana 2023) of the value function to facilitate the optimization over the Bellman Equation.

3. Dynamic Problems in Transportation

There is a broad range of dynamic problems in transportation. For an in-depth discussion, we refer to Soeffker, Ulmer, and Mattfeld (2022). The individual problems differ in their objective, constraints, the uncertain information, and the complexity of dynamic decisions. The latter is of particular interest for this work. Thus, instead of examining specific individual problems, we focus on three representative problem classes that differ in their decision space complexity. More specific, we differentiate between long-haul, medium-haul, and short-haul transportation problems. In the following, we briefly summarize the three classes and distill the core decision components in these problems.

Long-haul transportation usually comprises transportation of large batches from an origin to a destination and is done by truck, train, plane, or vessel. A classical example for long-haul transportation is the transport of goods between two distant cities via large trucks. Medium-haul describes the transportation of smaller entities in a confined region. A classical example is less-than-truckload routing where within the day a number of business customers (stores, factories, etc.) is served by a larger delivery truck that starts and ends its tour in a depot. Short-haul transportation requires the transportation of smaller entities to end-customers in a smaller region, often the infamous “last-mile” within the city and within a specified time-window. An example is attended home delivery of white goods where customers are present during the time-window to receive the delivery.

The main source of uncertainty for such applications is the uncertainty in demand (Soeffker, Ulmer, and Mattfeld 2022) and one main decision component is demand management, i.e., the selection of demand to serve (Fleckenstein, Klein, and Steinhardt 2023). For example, many providers of long-haul, medium-haul, or short-haul transportation receive requests during the day or operate on online freight exchange platforms (Miller, Nie, and Liu 2020). Such platforms match real-time transportation requests with transportation service providers. In both cases, in a first (capture) phase, transportation requests reveal over time. Requests are unknown until they are placed and are usually associated with a compensation payment, with shipment sizes, individual delivery locations (usually only medium- and short-haul), and with a delivery time-window (usually only short-haul). Providers dynamically select batches of requests they are willing to satisfy given their limited fleet resources. The fleet resources comprise the vehicle capacity and the limited routing time to visit the locations (only medium- and short-haul) within their time-windows

(only short-haul). After the first phase terminates, the selected requests are then satisfied in a second (fulfillment) phase. The goal of the providers is to use their limited resources effectively, i.e., to generate as much revenue as possible.

3.1. Example

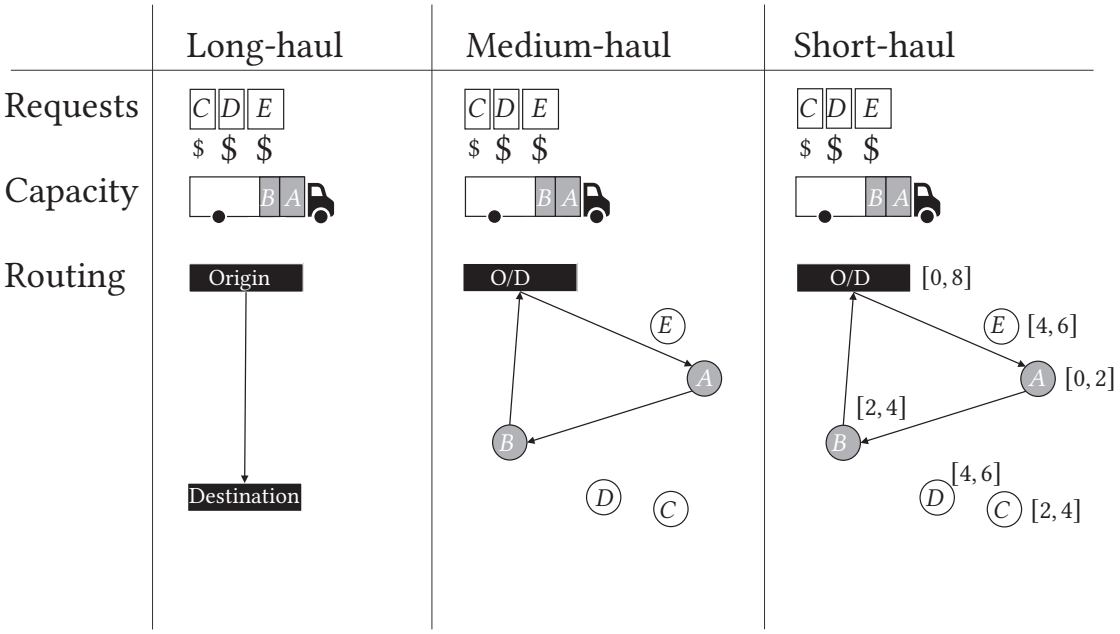
In the following, we illustrate the three problems via an example. Figure 1 depicts a decision state for each of the three problems. The left-most portion of the figure shows the example for long-haul transportation. In the state, two requests (A and B) were already selected and added to the vehicle (represented by the grey boxes). Three new requests (C, D, and E) are available for selection, represented by the white boxes. The size of the boxes indicate the required capacity. The size of the dollar symbol represents the compensation payment. For example, requests C and D have the same size but D provides a higher payment; requests D and E provide the same payment, but E requires more capacity. Long-haul transportation is performed from one predefined origin to one destination, thus, there is no routing involved. The decision for long-haul problems is now what requests to select given the limited capacity of the truck (and in anticipation of future requests before departure). Thus, the long-haul problem can be described as a dynamic knapsack problem (dKP).

The medium-haul problem is depicted in the middle portion. It is an extension of the long-haul problem. Now, routing decisions through the individual request locations (circles) must be made to ensure feasibility of the selection. This increases complexity of decision making. For example, while for the knapsack problem request D dominated request E, this is not the case for the medium-haul problem where request E is very close to previously selected request A. Furthermore, there is consolidation potential dependent on the selected subsets, e.g., if requests D is selected, request C does not require much additional routing time. In essence, the medium-haul problem extends the dKP by a routing component and maximum route length constrains. It can be represented by a dynamic capacitated orienteering problem (dCOP).

On the right side of the figure, the short-haul state is depicted. It is an extension of the medium-haul state as now, each request is associated with a delivery time window, indicated by the boxes next to the locations. This again leads to an increase in complexity in making effective decisions. For example, request C provides less payment than D but can be served directly after A without the vehicle idling. The final problem extends the dCOP to the dynamic capacitated orienteering problem with time windows (dCOPTW).

We note that while we focus on general long-haul, medium-haul, and short-haul transportation, the three components of limited capacity, routing constraints, and TWs can be found as essential parts of many other dynamic transportation applications, e.g. mobility on-demand (Kullman et al. 2022, Xie, Liu, and Chen 2023, Haferkamp, Ulmer, and Ehmke 2023, Heitmann et al. 2023), on-demand delivery (Ulmer and Thomas 2018, Voccia, Campbell, and Thomas 2019, Ulmer 2020, Dayarian, Savelsbergh, and Clarke 2020), or attended home grocery delivery (Campbell and Savelsbergh 2006, Agatz et al. 2011, Yang and Strauss 2017, Waßmuth et al. 2023).

Figure 1 **Example for the three problems.**



3.2. Problem Definition

Formally, the three problems are characterized as follows. A decision maker is presented a new set of requests I_k in each decision point $k = 1, \dots, K$. Requests $i \in I_k$ are characterized by their capacity requirement $w_{ki} \in \mathbb{R}^m$, their value $p_{ki} \in \mathbb{R}$, their location $l_{ki} \in A \subset \mathbb{R}^2$ (in the dCOP and dCOPTW only), and their time window $t_{ki} \in \mathbb{R}^2$ (in the dCOPTW only). The decision maker may choose requests from the set of presented requests. However, the set of requests chosen in this decision point and previous decision points must not exceed a given knapsack capacity, they must be served within a maximum tour length (only in the dCOP and dCOPTW), and must be served within their respective time windows (only in the dCOPTW). Note, that the routes are executed only after the end of the decision horizon. Once an request is chosen, it may not be discarded in later decision points and requests that have been rejected in previous decision points cannot be reconsidered in subsequent decision points. The objective is to maximize the cumulative value of requests collected over all decision points.

In the following, we model the three problems as sequential decision processes. Sequential decision processes are modeled by a sequence of *decision points* $k = 1, \dots, K$. In each decision point we derive a *decision* x_k based on the current *state* S_k. The state S_k is given by the known information relevant to decision-making. After a decision is made, a *reward* $R(S_k, x_k)$ is assigned based on the state and the decision. Finally, we *transition* $S^M : (S_k, x_k, W_{k+1}) \mapsto S_{k+1}$ to the next state S_{k+1} based on the current state S_k, the decision taken x_k, and the realization of *stochastic information* W_{k+1} sampled from an exogenous process. We first

define the SDP for the dKP in Subsection 3.3 before we modify relevant elements of the SDP to match the dCOP in Subsection 3.4 and dCOPTW in Subsection 3.5.

3.3. Dynamic Knapsack Problem

The dKP can be defined as follows:

Decision point: A decision point $k \in \{1, \dots, K\}$ occurs when a new set of requests is revealed. The number of decision points K is deterministic.

State: A state $S_k = (k, b_k, w_k, p_k)$ in decision point k consists of the current decision point k, the remaining knapsack capacity b_k, and the capacity requirements w_{ki} and values p_{ki} of all requests $i \in I_k$ revealed in decision point k. The number of revealed requests is given by $n = |I_k|$. We denote the space of all states as $\mathcal{S}$.

Decision: A decision $x_k \in \{0,1\}^n$ defines which of the n requests in I_k are chosen. We denote the space of all feasible decisions in state S_k as $\mathcal{X}(S_k)$. The decision space is characterized by the following constraints

$$\sum_{i \in I_k} w_{ki} \cdot x_{ki} \leq b_k \tag{2a}$$

$$x_{ki} \in \{0,1\} \quad \forall i \in I_k \tag{2b}$$

Decisions are derived by a policy $\pi : \mathcal{S} \to \mathcal{X}$ that assigns a decision x_k to each given state S_k. We denote the optimal policy as π^*.

Reward: The reward function $R : \mathcal{S} \times \mathcal{X} \to \mathbb{R}$ assigns a reward $R(S_k, x_k)$ to each state-decision tuple. The Reward $R(S_k, x_k)$ in state S_k is given by the inner product of reward vector p_k and the decision taken x_k:

$$R(S_k, x_k) = p_k^\intercal x_k \tag{3}$$

Post-decision state: The post-decision state $S_k^x := (k+1, b_{k+1})$ represents the state immediately after a decision was made but before new information is revealed. It consists of the updated decision point $k+1$ and the updated capacity $b_{k+1} = b_k - w_k^\intercal x_k$. We denote the space of post decision states as $\mathcal{S}^x$.

Stochastic information: The stochastic information $W_{k+1} \in \Omega$ is revealed after the decision x_k is made in state S_k. The stochastic information consists of the new set of requests I_{k+1}.

Transition: The transition $S^M : \mathcal{S}^x \times \Omega \to \mathcal{S}$ maps a post-decision state and a stochastic information to a new state $S_{k+1} = S^M(S_k^x, x_k, W_{k+1})$.

Value function: The value function $V : \mathcal{S}^x \to \mathbb{R}$ assigns each post-decision state S_k^x the expected reward-to-go if we follow the optimal policy π^*:

$$V(S_k^x) = \mathbb{E}\Big[\sum_{l=k+1}^{K} R\big(S_l, \pi^*(S_l)\big) \mid S_k^x \Big] \tag{4}$$

Objective function: The objective is to identify a policy that maximizes the expected total reward over all the decision points:

$$\max_{\pi \in \Pi} V^{\pi}(S_0). \tag{5}$$

3.4. Dynamic Capacitated Orienteering Problem

The dCOP extends the dKP by a spatial component as requests $i \in I_k$ are now also associated with a location $l_{ki} \in A$ in a service area $A \subset \mathbb{R}^2$. Each pair of locations corresponding to requests i and j is associated with a distance $d_{ij} \in \mathbb{R}_+$. If a request is chosen, its location must be visited within a tour that starts at the depot, ends at the same depot, and visits all requests chosen prior. The tour must not exceed a maximal tour length L. However, the tour is tentative and can be altered in every decision point, i.e., it is only constructed to assert the feasibility of a decision. The actual tour is only executed after the end of the decision horizon. Thus, a decision's feasibility depends on the requests chosen in previous decision steps. For that reason, we introduce the index set $\bar{I}_k$ denoting the indices of all requests chosen before decision point k. We extend our description of states and decisions accordingly:

State: A state $S_k = (b_k, w_k, p_k, l_k)$, is now extended by the locations l_{ki} with $i \in \bar{I}_k \cup I_k$, i.e., the locations $l_{ki}, i \in \bar{I}_k$ that were chosen in the previous decision points and the newly revealed locations $l_{ki}, i \in I_k$.

Decision: A decision $x_k \in \{0,1\}^n$ describes which requests in I_k are chosen. We further define $x_{ki} = 1,\ \forall i \in \bar{I}_k$. The decision x_k must not only fulfill capacity constraints but there also must exist a tour of length lesser than or equal to the maximum tour length L that visits all requests previously and currently chosen. We denote the depot at the start of the tour as 0 and the (virtual) depot at the end of the tour as $N = 1 + \sum_{l=0}^{K} |I_l|$. Further, we denote the set of all known requests as $I := \bar{I}_k \cup I_k$ and $I_+ = I \cup \{0, N\}$. We introduce the decision variable y_{ij} denoting if the tour contains an arc from request i to request j and the decision variable s_i denoting when the tour arrives at location $i \in I_+$. Finally, we characterize the decision space by the following set of constraints:

$$\sum_{i \in I_k} x_{ki} w_{ki} \leq b_k \tag{6a}$$

$$\sum_{i \in I_+} y_{0,j} = \sum_{i \in I_+} y_{i,N} = 1 \tag{6b}$$

$$\sum_{i \in I_+} y_{im} - \sum_{j \in I_+} y_{mj} = 0 \qquad m \in I \tag{6c}$$

$$\sum_{i \in I} y_{ij} = x_{kj} \qquad j \in \bar{I}_k \tag{6d}$$

$$s_i + d_{ij} - s_j \leq L \cdot (1 - y_{ij}) \qquad i \in I, j \in I \tag{6e}$$

$$s_{n \cdot k+1} \leq L \tag{6f}$$

$$x_{ki} = 1 \qquad i \in \bar{I}_k \tag{6g}$$

$$x_{ki} \in \{0,1\} \qquad i \in I_k \tag{6h}$$

$$y_{ij} \in \{0,1\} \qquad i \in I_+, j \in I_+ \tag{6i}$$

$$s_i \in [0, L] \qquad i \in I_+ \tag{6j}$$

Constraint (6a) ensures that the knapsack capacity is not exceeded. Constraint (6b) guarantee that the tour starts and ends at the depot. Constraints (6c) provide the connectivity of the tour. Constraints (6d) state that every chosen request must be visited and discarded requests cannot be visited. Constraints (6e) determine the timeline of the tour. Constraint (6f) ensures that the maximum tour length is not exceeded. Constraints (6g) specify that previously selected requests must also be selected in this decision point.

3.5. Dynamic Capacitated Orienteering Problem with Time Windows

The dCOPTW further extends the dCOP by a temporal component. Each request $i \in I_k$ is now also associated with a time window $t_{ki} := (t_{ki}^-, t_{ki}^+)$. We update state- and decision space accordingly:

State: A state $S_k = (b_k, w_k, p_k, l_k, t_k)$, is now additionally given by the time windows t_{ki} with $i \in I$.

Decision: The decision space is still characterized by Constraint (6a)-(6g) as well as by the following time window constraints:

$$t_{ki}^- \leq s_i \leq t_{ki}^+ \quad \forall i \in I \tag{7a}$$

Constraints (7a) ensure that the lower and upper bounds of the time windows are met.

4. Methodology

We now turn towards solving the described sequential decision processes. As already illustrated with the small example in Figure 1, finding optimal policies is intractable because (1) the decision space is complex and searching it requires significant time, and (2) the value of a decision is unknown as it depends on future information and decisions (i.e., the future value). Instead, we employ reinforcement learning strategies that search the complex space and use an approximating of the future value of a decision in a state.

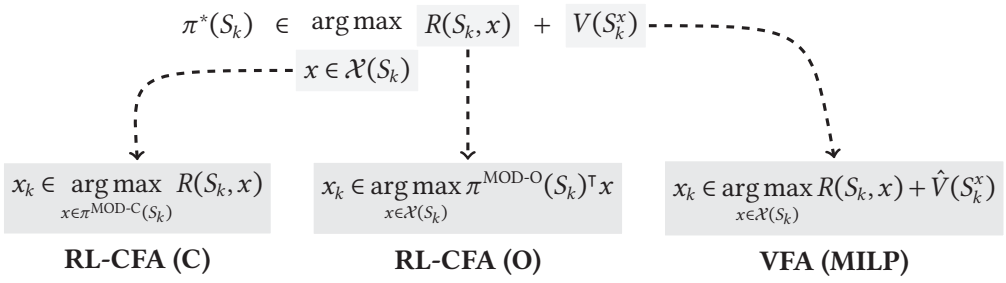
In Subsection 4.1, we discuss how optimal solutions can be found in theory and motivate based on the Bellman Equation the selection of our methodology. In Subsection 4.2, we give an example for the general functionality of our methods and then provide the algorithmic details for the VFA (MILP) Subsection 4.2.1) and the RL-CFA methods Subsection 4.2.2).

4.1. The Bellman Equation

To characterize an optimal solution, i.e., an optimal policy π^*, for a given sequential decision process, we require the notion of a value function V^π. The value function V^π corresponds to the value of being in a state $S_k \in \mathcal{S}$ and following a policy $\pi \in \Pi$. It is formally defined as

$$V^\pi(S_k) = \mathbb{E}_\pi \Big[\sum_{t=k}^{K} R\big(S_t, \pi(S_t)\big) \mid S_k \Big]. \tag{8}$$

Figure 2 **Three policies derived from the Bellman Equation** (11).



These value functions establish a partial ordering over the domain of policies Π: A policy π is better or equal than a policy π', i.e, $\pi \geq \pi'$, if $V^\pi(S_k) \geq V^{\pi'}(S_k)$ for all $S_k \in \mathcal{S}$. On this basis, we may characterize the set of optimal policies $\Pi^* \subset \Pi$ by $\pi^* \in \Pi^*$ if and only if $\pi^* \geq \pi, \forall \pi \in \Pi$ (Sutton and Barto 2018). Consequently, there is also the *optimal* value function

$$V(S_k) = \max_{\pi \in \Pi} V^\pi(S_k), \quad \forall S_k \in \mathcal{S} \tag{9}$$

The optimal value function can also be defined recursively according to

$$V(S_k) = \max_{x \in \mathcal{X}(S_k)} R(S_k, x) + \mathbb{E}[V(S_{k+1}) \mid S_k^x]. \tag{10a}$$

$$\mathbb{E}[V(S_{K+1}) \mid S_K^x] = 0, \tag{10b}$$

for all terminal post-decision states S_K^x. In the rest of this work, and by a slight abuse of notation, we refer to the optimal value function only as value function and we notate the *reward-to-go* $\mathbb{E}[V(S_K) \mid S_K^x]$ as $V(S_k^x)$.

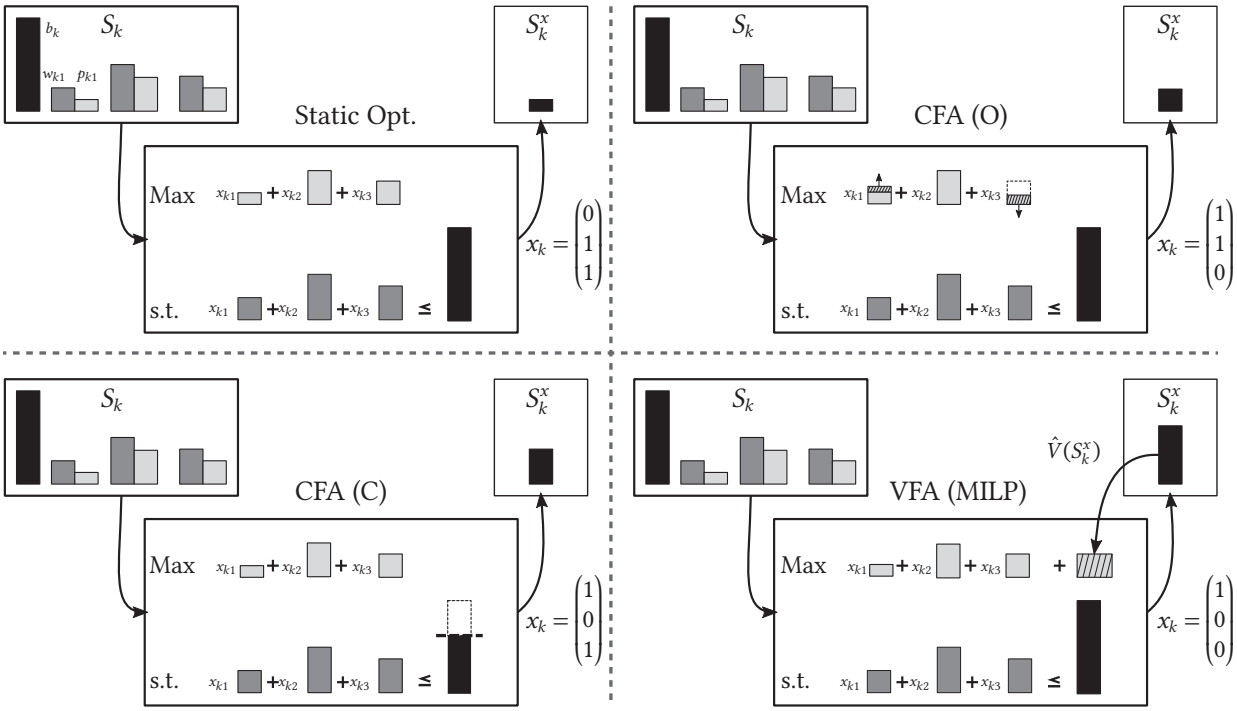
By this recursive definition, an optimal policy $\pi^* \in \Pi^*$ satisfies the Bellman Equation, i.e., it maximizes the immediate reward plus the expected reward-to-go over all the decision points

$$\pi^*(S_k) \in \arg\max_{x \in \mathcal{X}(S_k)} R(S_k, x) + V(S_k^x). \tag{11}$$

Unfortunately, Equation (11) is almost never tractable due to the *Curses of Dimensionality* (see Powell 2022). Therefore, approximate solution methods, such as the ones discussed in Section 1, are often successfully applied. In this work, we extend two famous policies for sequential decision processes, namely cost function approximation (CFA) and value function approximation (VFA) methods. Our proposed policies are derived directly from each of the three parts of the Bellman Equation (11) as highlighted in Figure 2.

The three parts of the Bellman Equation that we are referring to are given by the decision space $\mathcal{X}(S_k)$, the reward function $R(S_k, x)$, and the value function or reward-to-go $V(S_k^x)$. Our first policy, *RL-CFA (C)*, manipulates the decision space based on current state information before maximizing the current reward over the manipulated decision space. Our second policy, *RL-CFA (O)*, manipulates the current reward associated with each decision before maximizing the manipulated current reward over the entire decision space. Our third policy, *VFA (MILP)*, approximates the value function by a piecewise affine neural network VFA before maximizing the current reward plus the approximated reward-to-go over the entire decision space. Optimizing over each decision space is performed by dedicated MILP solvers for all three policies.

Figure 3 Overview of our solution methods at the example of the dKP.



4.2. Solution Methods

We provide an intuition for our proposed methods at the hand of an introductory example, highlighted in Figure 3, before we describe them in detail in Subsection 4.2.1 and Subsection 4.2.2.

Figure 3 is divided into four parts. Each part shows the same decision state in the dynamic Knapsack problem but corresponds to a different solution method, namely a static optimization policy (Static Optimization, *top, left*), a manipulation of the MILP by altering p_k (RL-CFA O, *top, right*), a manipulation of the MILP by altering b_k (RL-CFA C, *bottom, left*), and an extension of the MILP's objective function by a value function approximation (VFA MILP, *bottom, right*). The decision state S_k in the dynamic Knapsack problem consists of $n = 3$ presented requests. The items are represented by their weights w_k, depicted in dark gray, and their values p_k, shown in light gray. For the sake of example, we assume that the weights of the presented requests are given by $w_k = (2, 4, 3)^\intercal$ and their values are given by $p_k = (2, 4, 3)^\intercal$. Further, the state includes the current Knapsack capacity, given by the black bar in the state S_k and on the right side of the constraints. We assume that the current knapsack capacity has a numerical value of $b_k = 8$. Next, we highlight how each policy derives a decision x_k from the state S_k.

Static Optimization. The static optimization policy (*top, left*) maximizes the immediate reward such that the sum of weights does not exceed the knapsack capacity of $b_k = 8$. This is achieved by taking request two and three $x_k = (0, 1, 1)^\intercal$, collecting a reward of $R_k = p_2 + p_3 = 4 + 3 = 7$, and reducing the knapsack capacity to $b_{k+1} = b_k - w_2 - w_3 = 8 - 4 - 3 = 1$.

RL-CFA (O). The RL-CFA (O) policy (*top, right*) manipulates the request values p_k in order to maximize not only the immediate reward but also the reward-to-go. This policy derives decisions on the current state information but does not explicitly plan into the future. Instead, it learns behaviors that tend to do well with regard to future uncertainty. In our example, the policy might update the request values to $p'_k = (2, 3, -1)$. In the next step, the virtual immediate reward is maximized such that the capacity constraints are met. Thus, requests one and two are chosen $x_k = (1, 1, 0)^\intercal$, resulting in a virtual reward of $R'_k = p'_1 + p'_2 = 2 + 3 = 5$ (and an actual reward of $R_k = 2 + 4 = 6$), and updating the capacity to $b_{k+1} = b_k - w_1 - w_2 = 8 - 2 - 4 = 2$.

RL-CFA (C). The RL-CFA (C) policy (*bottom, left*) works analogously to the second policy, but manipulates the current capacity b_k by virtually decreasing it. Again, the decision is based on current state information but the policy does not explicitly plan into the future. It learns behaviors that work well with regard to future uncertainty. In our example, the policy might modify the current knapsack capacity to $b'_k = 5$ in order to reserve some capacity for later decision points. We then maximize the immediate reward such that the modified capacity b'_k is not exceeded. This results in taking request one and three ($x_k = (1, 0, 1)^\intercal$), collecting a reward of $R_k = p_1 + p_3 = 2 + 3 = 5$, and updating the knapsack capacity to $b_{k+1} = b_k - w_1 - w_3 = 8 - 2 - 3 = 3$.

VFA (MILP). Our final policy (*bottom, right*), aims to maximize the current reward plus an expected future reward given by a value function approximation of the post-decision state (here, the post-decision state is given by the time and the updated knapsack capacity b_{k+1}). Thus, the policy plans explicitly into the future. In our example, the value function approximation could take the form of a linear mapping

$$\hat{V}(S_k^x) = 1.1 \cdot (b_k - w_k^\intercal x_k).$$

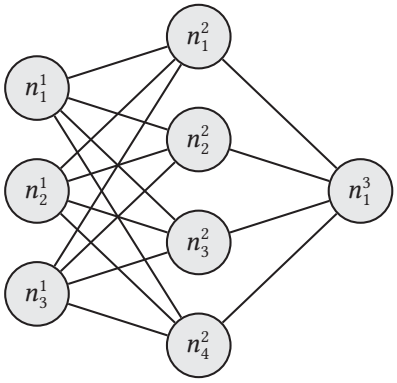
Note that this linear mapping is just an arbitrary example and we will use more expressive neural networks in our actual implementation. Maximizing the immediate reward plus the expected reward-to-go according to $\hat{V}$ leads to the first request being selected $x_k = (1, 0, 0)^\intercal$, resulting in an immediate reward of $R_k = p_1 = 2$ (and an expected future reward of $\hat{V}(b_{k+1}) = \hat{V}(6) = 6.6$), and an updated capacity of $b_{k+1} = b_k - w_1 = 8 - 2 = 6$.

Next, we detail how concrete policies can be derived from these classes for our benchmark problems.

4.2.1. Extending the Objective Function of the MILP. An optimal decision policy *per definitionem* solves each decision state for the decision that maximizes the current reward plus the expected reward-to-go conditioned on the corresponding post-decision state:

$$x_k \in \underset{x_k \in \mathcal{X}(S_k)}{\arg\max}\, p_k^\intercal x_k + V(S_k^x). \tag{12}$$

Figure 4 Application of the notation to a small neural network.



Here, the expected reward-to-go is represented by the value function $V(S_k^x)$ of the post-decision state S_k^x if we take decision x_k in state S_k. Of course, V is unknown but we may approximate it with the help of a predictive model $\hat{V}_\omega$ with trainable weights ω. Such a model maps a post-decision state S_k^x to a scalar approximate $v_k^x := \hat{V}_\omega(S_k)$ of the expected reward-to-go $V(S_k^x)$. Further, the predictive model should be sufficiently expressive to approximate arbitrary value functions as well as result in a benign optimization problem when substituting $V(S_k^x)$ by $v_k^x := \hat{V}_\omega(S_k^x)$ in Equation (12) (see also Figure 3, *bottom, right*).
We choose a standard feed-forward neural network with ReLU-activations, $ReLU(v) = \max(v, 0)$, as the predictive model. Such a network is expressive enough to approximate any (potentially non-linear) value function while being piecewise affine. The neural network is given by its $l \in \mathbb{N}$ layers and corresponding number of neurons m_i, $i \in \{1, \dots, l\}$ in layer i. Each layer $i+1$ can be recursively defined by the output n^i of the previous layer's nodes $\{n_j^i\}$, $j = 1, ..., m_i$. In the Figure 4, we highlight how the definition of each node n_j^i corresponds to a neural network with 3 layers and 3,4, and 1 nodes per layer.

Formally, we define the output of neuron j in layer i as

$$n_j^i := \mathrm{ReLU}\left(\omega^{i,j} \cdot n^{i-1} + \beta^{i,j}\right). \tag{13}$$

Here, $\omega := \{\omega^{i,j}, \beta^{i,j} \mid i \in \{1, \dots, l\}, j \in \{1, \dots, m_i\}\}$ denotes the trainable weights and biases of the neural network. The ReLU activations can be written as big-M constraints (Anderson et al. 2020). While stronger formulations for ReLU activations over an affine function exist, big-M formulations perform well for small neural networks (Anderson et al. 2020). Let M_{ij}^+ and M_{ij}^- denote the maximum and minimum value that our affine function $g(v) = \omega^{i,j} \cdot v + \beta^{i,j}$ attains over its input domain. Then, we write the output of neuron j in layer i as

$$n_j^i \geq \omega^{i,j} \cdot n^{i-1} + \beta^{i,j} \tag{14a}$$

$$n_j^i \leq \omega^{i,j} \cdot n^{i-1} + \beta^{i,j} - M_{ij}^- \cdot (1 - z) \tag{14b}$$

$$n_j^i \leq M_{ij}^+ \cdot z \tag{14c}$$

$$z \in \{0, 1\} \tag{14d}$$

By expressing the neural network VFA $\hat{V}_\omega$ as a set of linear constraints, we define our policy π as follows:

$$\pi(S_k) = \underset{x_k \in \mathcal{X}(S_k)}{\arg\max}\, p_k^\intercal x_k + \hat{V}(S_k^x). \tag{15}$$

We acknowledge that this mapping is not well-defined in general as the problem might not be feasible or it might correspond to a one-to-many mapping. However, under the assumption off a non-empty decision space ($x_k = \vec{0}$ is always feasible) and since solving Equation (15) to optimality using a standard solver returns a single maximizer, this notation is reasonable.

In practice, it is often not desirable to use the entire post-decision state S_k^x as input of $\hat{V}$ due to its high dimension (Soeffker, Ulmer, and Mattfeld 2019). Instead, we map a vector of post-decision state features $f_k^x := \phi(S_k^x)$ to a scalar approximate $v_k := \hat{V}_\omega(f_k^x)$ of the expected reward-to-go $V(S_k^x)$. In the following, we show how an implementation of the decision policy may look like for our stochastic dynamic transportation problems. For ease of exposition, we assume our value function has one hidden layer with m neurons, our input features are given as f_k, and the final output of the neural network v_k^x is scalar. Then, we may extend the objective function by our approximation of the expected reward-to-go as given in Model (16).

$$\max \quad \left[\sum_{i=0}^{n-1} p_{ki} x_{ki}\right] + v_k^x \tag{16a}$$

$$\text{s.t.} \quad n_j^1 \geq \omega^{1,j} f_k^x + \beta^{1,j} \qquad j = 1, \dots, m \tag{16b}$$

$$n_j^1 \leq \omega^{1,j} f_k^x + \beta^{1,j} - M_{1j}^-(1-z) \qquad j = 1, \dots, m \tag{16c}$$

$$n_j^1 \leq M_{1j}^+ z \qquad j = 1, \dots, m \tag{16d}$$

$$v_k^x = \omega^{2,1} n^1 + \beta^{2,1} \tag{16e}$$

$$z \in \{0, 1\} \tag{16f}$$

$$x_k \in \mathcal{X}(S_k) \tag{16g}$$

We omitted the definition of feature vector f_k^x in Model (16). Since the feature vectors f_k^x should depend on the current decision variables in the respective MILP, they must be defined in a way tractable for the MILP. For that reason, we choose the mapping ϕ to be linear.

We construct the feature vectors f_k for the dKP, dCOP, and dCOPTW as follows. For the dKP, a post-decision state is given by the current time of the decision state and the remaining knapsack capacity after taking a decision x_k. Thus, we may define $f_k \in \mathbb{R}^2$ as

$$f_{k1}^x = t_k \tag{17a}$$

$$f_{k2}^x = b_k - \sum_{i=0}^{n-1} x_{ki} p_{ki} \tag{17b}$$

For the dCOP, we additionally provide the current route length to characterize the decision state:

$$f^x_{k3} = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} y_{kij} d_{kij} \tag{18a}$$

For the dCOPTW, we further include the length of the subroutes induced by each set of time windows $t^1, \dots, t^M$ (we assume M uniform, disjunct time windows)

$$f^x_{k,3+m} = \sum_{i \in I(t^m)} \sum_{j=0}^{n-1} y_{kij} d_{kij} \quad \forall m = 1, \dots, M \tag{19a}$$

with $I(t^m) := \{i \in I \mid t_{ki} = t^m\}$. By combining the problem constraints, the neural network constraints, and the feature constraints, we may solve for a feasible decision that maximizes current reward plus the expected future reward according to our neural network value function approximation. We refer to Appendix C for our training procedure and related implementation details.

4.2.2. Dynamically Manipulating the MILP. A potential drawback of integrating a VFA directly into a MILP is the added complexity introduced into the MILP, as well as the fact that we now directly search over all possible approximation errors of the VFA in the MILP. In our RL-CFA (C) and RL-CFA (O) approaches, we manipulate the constraints and reward vector of the MILP, respectively, depending on current state information before we solve the MILP. The RL-CFA (C) policy reserves capacities for later decision points, i.e., by decreasing the knapsack capacity or maximum route length as shown in Figure 3 (*bottom, left*). The RL-CFA (O) policy alters the rewards associated with each request to incentivize or disincentivize choosing certain requests as shown in Figure 3 (*top, right*).

The corresponding decision policy $\pi : \mathcal{S} \to \mathcal{X}$ for the sequential decision problem is given by first manipulating the reward or decision space based on the state S_k before choosing a decision according to the static optimization policy. (Policy (20a) and policy (20b) are well-defined if their decision spaces are non-empty and if we use a recovery procedure that returns a unique maximizer instead of the entire set of maximizers). This may result in one of the two proposed policies:

$$\pi(S_k) = \arg\max_{x \in \mathcal{X}(S_k)} \pi^{\text{MOD-O}}(S_k)^\intercal x \tag{20a}$$

$$\pi(S_k) = \arg\max_{x \in \pi^{\text{MOD-C}}(S_k)} R(S_k, x) \tag{20b}$$

Here, the manipulation of the reward, RL-CFA (O), is of the form $\pi^{\text{MOD-O}} : \mathcal{S} \to \mathbb{R}^n$ and the manipulation of the constraints, RL-CFA (C), is of the form $\pi^{\text{MOD-C}} : \mathcal{S} \to \mathcal{P}(\mathcal{X})$. $\pi^{\text{MOD-O}}$ manipulates the reward p_k of requests according to

$$p'_k = \pi^{\text{MOD-O}}(S_k) \tag{21}$$

with the image of $\pi^{\text{MOD-O}}(\mathcal{S})$ being chosen such that negative request values may occur to ensure that such requests are not chosen. The second manipulation, $\pi^{\text{MOD-C}}$, restricts the knapsack capacity b_k (as well as

the maximum tour length L in the case of the dCOP and dCOPTW). Technically, the manipulation maps a state to a subset of the decision space $\mathcal{X}(S_k)$. However, as the restricted decision space is induced by the modification of the capacities b_k and L_k, we do not explicitly construct the restricted decision space. Instead, the knapsack capacity b_k is directly modified according to

$$b'_k = b_k \cdot \pi^{\text{MOD-C}}(S_k) \tag{22}$$

with the image of $\pi^{\text{MOD-C}}(\mathcal{S})$ being contained in $[0, 1]$ to ensure that the updated capacity is lesser equal than the true capacity but remains positive. In the case of dCOP and dCOPTW, we additionally modify the maximal route length L by

$$L' = L \cdot \pi^{\text{MOD-C}}(S_k) + L_{k-1} \cdot (1 - \pi^{\text{MOD-C}}(S_k)) \tag{23}$$

with the image of $\pi^{\text{MOD-C}}(\mathcal{S})$ again being contained in $[0, 1]$ to ensure that the previous route with length L_{k-1} is still feasible. For the dCOP and dCOPTW, we learn the manipulation of b and L in a single policy $\pi^{\text{MOD-C}} : \mathcal{S} \to [0, 1]^2$. In our implementation, the manipulation policies have access to the full state information, i.e., no information is aggregated.

The manipulations $\pi^{\text{MOD-C}}$ and $\pi^{\text{MOD-O}}$ can be learned by any reinforcement learning method that allows for continuous decisions. We employ a deep deterministic policy gradient method (DDPG) (Lillicrap et al. 2015). In such an implementation, the policy π^{MOD} is learned by a neural network π^{MOD}_θ with trainable weights θ. π^{MOD}_θ is updated with the help of a second, independent Q-network Q_ω with trainable weights ω. Q serves as an estimator of the immediate reward plus reward-to-go given a state-decision tuple, i.e., $Q_\omega : \mathcal{S} \times \mathcal{X} \to \mathbb{R}$. We provide details on the training framework in Appendix C.

4.3. Policy Discussion

While we focus on individual problems, the proposed policies are general. In this section, we show that under reasonable assumptions, the proposed policy classes can emulate any deterministic policy. In the case of the VFA integrated in the MILP, we further sketch conditions in which we can show convergence guarantees. For the rest of this section, we make the following assumption that are in line with most transportation problems, including our proposed problem classes.

Assumption 1. *We assume the sequential decision process to have a finite number of decision points, i.e., $k \leq K \in \mathbb{N}$. We further assume that for all $S_k \in \mathcal{S}$ the decision space $\mathcal{X}(S_k)$ is non-empty and can be expressed as a set of linear constraints as follows*

$$\mathcal{X}(S_k) = \{x_k \in \mathbb{Z}^n_+ \mid \exists v \in \mathbb{R}^{m-n}_+ \text{ s.t. } A_k[x_k, v]^T \leq b_k\}, \tag{24}$$

with $A_k \in \mathbb{R}^{m \times n}$, $b_k \in \mathbb{R}^m$ being a state-dependent matrix and vector, respectively, and $v \in \mathbb{R}^{n-m}_+$ being a set of positive continuous decision variables. We further assume that the reward R_k in a decision point k is bounded and is given by the inner product of the decision $x_k \in \mathcal{X}(S_K)$ and the profit vector $p_k \in \mathbb{R}^n$, i.e., $R_k = p_k^\intercal x_k$.

While we already introduced our policy classes for our benchmark problems, we now formally define our proposed classes for an even more general setting. Our first policy class *RL-CFA (O)* manipulates the value of decisions based on the current state information.

Definition 1 (RL-CFA O). We call a policy $\pi \in \Pi$ a *RL-CFA (O)* policy, if there exists a mapping $\pi^{\text{MOD-O}}(S_k) : \mathcal{S} \to \mathbb{R}^n$ such that π satisfies

$$\pi(S_k) \in \underset{x_k \in \mathcal{X}(S_k)}{\arg\max}\, \pi^{\text{MOD-O}}(S_k)^\intercal x_k, \quad \forall S_k \in \mathcal{S}$$

Our second policy class, manipulates the decision space $\mathcal{X}(S_k)$ in order to save resources for later decision points.

Definition 2 (RL-CFA C). We call a policy $\pi \in \Pi$ a *RL-CFA (C)* policy, if there exists a mapping $\pi^{\text{MOD-b}} : \mathcal{S} \to \mathcal{P}(\mathcal{X})$ such that π satisfies

$$\pi(S_k) \in \underset{x_k \in \pi^{\text{MOD-b}}(S_k)}{\arg\max}\, p_k^\intercal x_k, \quad \forall S_k \in \mathcal{S}.$$

Our third policy class approximates the post-decision state value by a piecewise affine function and solves for a feasible decision that maximizes current plus approximated future reward.

Definition 3 (VFA MILP). We call a policy $\pi \in \Pi$ a *VFA (MILP)* policy, if there exists a bounded, piecewise affine mapping $\hat{V} : \mathcal{S}^x \to \mathbb{R}$ such that π satisfies

$$\pi(S_k) \in \underset{x_k \in \mathcal{X}(S_k)}{\arg\max}\, p_k^\intercal x_k + \hat{V}(S_k^x), \quad \forall S_k \in \mathcal{S}.$$

The proposed policy classes are highly flexible. In fact, the policies can emulate any deterministic policy, i.e., also the optimal policy, under reasonable assumptions.

Proposition 1. *Let Π_D denote the space of deterministic policies for a sequential decision process that satisfies Assumption* (1)*. Further, let Π_O, Π_C, Π_V denote the space of* RL-CFA (O), RL-CFA (C) *and* VFA (MILP) *policies for this sequential decision process, respectively. Then it holds that*

$$\Pi_D = \Pi_C = \Pi_V. \tag{25}$$

If we require binary decision vectors in every decision point $\mathcal{X}(S) \subset \{0,1\}^n$, then it holds that

$$\Pi_D = \Pi_O = \Pi_C = \Pi_V \tag{26}$$

Proof. See Appendix A.

Typically, we do not want to emulate any given policy but rather approximate the unknown optimal policy. Thus it is important to characterize when our policies coincide with the optimal policy.

Proposition 2. *Let* $\{\hat{V}_i\}_{i\in\mathbb{N}}$ *be a sequence of value function approximations with*

$$\hat{V}_i : \mathcal{S}^x \to \mathbb{R}, \quad \forall i \in \mathbb{N}$$

and let $\{\pi_i\}_{i\in\mathbb{N}} \subset \Pi_V$ *be a corresponding sequence of VFA (MILP) policies satisfying*

$$\pi_i(S_k) \in \underset{x_k\in\mathcal{X}(S_k)}{\arg\max}\, p_k^\intercal x_k + \hat{V}_i(S_k^x), \quad \forall S_k \in \mathcal{S}.$$

If $\lim_{i\to\infty} \hat{V}_i = V$ *uniformly, then* $\bar{\pi} := \lim_{i\to\infty} \pi_i$ *is an optimal policy.*

Proof. See Appendix A.

Indeed, there are known conditions in which we can construct a sequence $\{\hat{V}_i\}_{i\in\mathbb{N}}$ which converges with probability one to V.

Proposition 3. *Given a sequential decision process with fine state space* $|\mathcal{S}| < \infty$*, finite decision space* $|\mathcal{X}| < \infty$*, and a bounded reward function R, and given a policy* π*, such that* $\mathbb{P}\left[x \in \mathcal{X}(S) \mid S \in \mathcal{S}\right] > 0$ *for all state-decision pairs* $(x, S) \in \mathcal{X} \times \mathcal{S}$*. Then, the sequence* $\{\hat{V}_i\}_{i\in\mathbb{N}}$ *defined by the update rule*

$$\hat{V}_{t+1}(S_t^x) = \hat{V}_t(S_t^x) + \alpha_t[R_t + \gamma \cdot \max_{x'\in\mathcal{X}(S_{t+1})} \hat{V}_k(S_{t+1}^{x'}) - \hat{V}(S_t^x)] \tag{27}$$

converges with probability one *to the optimal value function V for a sequence of states* $\{S_t\}_{t\in\mathbb{N}} \subset \mathcal{S}$ *visited by* π *if*

$$\sum_t \alpha_{t=1}^{\infty} = \infty \qquad \sum_{t=1}^{\infty} \alpha^2 < \infty \tag{28}$$

for all $S_t^x \in \mathcal{S}^x$.

Proof. See the convergence proof of the standard Q-learning technique (Fan et al. 2020).

Having detailed our proposed policy classes, we apply them to long-haul-, medium-haul-, and short-haul transportation.

5. Experimental Design

We first describe the design of our experiments and the benchmark policies before we present and analyze the results in Section 6. Our experiments are designed to analyze the given problems as well as the proposed policy classes. They extend well-known instances from the literature by common assumptions to cover capacity constraints (dKP), route length constraints (dCOP), and time-window constraints (dCOP). The parameterization of the instances is chosen such that capacity- and tour-length constraints are equally restricting, and such that the workload per vehicle is realistic for long-haul, medium-haul, and short-haul problems, respectively (5 to 25 customers are served per day and vehicle). We consider instances with large combinatorial decision spaces ($n > K$) to analyze the value of searching the decision space and instances with a long decision horizon ($K > n$) to analyze the value of evaluating decisions with

respect to future uncertainty. The rest of the section is structured as follows. First, we describe the general parameters of the experiments. Second, we define the request distributions (including weights, values, and vehicle capacities) that are shared by all three problems. Third, we define the spatial distributions of locations to visit, the location of the depot, and the maximum tour length. These characteristics are shared by the dCOP and dCOPTW. Fourth, we define how we sample time windows in the dCOPTW. Fifth, we define the performance metrics used in our computational study. Sixth, we describe our benchmarks methods.

General Design: For ease of exposition, we consider a deterministic number of equidistant decision points. The number of requests per decision point is also deterministic and equal in all decision points. We conduct three sets of experiments. The first set of experiments considers $n = 3$ requests per decision point and $K = 5$ decision points. The second set of experiments analyzes the value of evaluating decisions with respect to future uncertainty and, therefore, considers $n = 3$ requests per decision point and $K = 15$ decision points. The third set of experiments analyzes the value of searching the decision space and, therefore, considers $n = 10$ requests per decision point and $K = 5$ decision points.

Item Distributions: We follow the work of Chu and Beasley (1998) to sample request weights and rewards, and to choose the knapsack capacity: All weights are sampled uniformly random from the half-open interval $[0, 1)$, i.e., $w_{ki} \sim U(0, 1)$, $\forall k = 1, \dots, K$, $\forall i = 1, \dots, n$. The initial knapsack capacity is set according to $b_1 = \alpha \sum_{k=1}^{K} \sum_{i=1}^{n} w_{ki}$. We choose a restrictive knapsack capacity of $\alpha = 0.3$ in all our experiments. The values of all requests are dependent on their weights and chosen according to $p_{ki} = w_{ki} + 0.5 \cdot q_{ki}$, $\forall k = 1, \dots, K$ and $\forall i = 1, \dots, n$ with $q_{ki} \sim U(0, 1)$ being a random variable following a uniformly random distribution in the half-open interval $[0, 1)$. In the case of the dCOP and dCOPTW, the rewards are independent of spatial characteristics (discussed in the following).

Spatial Distribution: We consider a square service area of unit size with centrally located depot. All locations l_{ki}, $\forall k = 1, \dots, K$, $\forall i = 1, \dots, n$ are uniformly random distributed within the service area A. The distance between two locations is given by their Euclidean distance. The maximum tour length is chosen according to $L = \alpha \cdot \sqrt{|A| \cdot n \cdot K}$ with $|A|$ being the surface area of the service area in the case of the dCOP and $L = \alpha \cdot \sqrt{|A| \cdot n \cdot K \cdot m}$ in the case of the dCOPTW (m being the number of time window slots). This continuous approximation is chosen such that in approximately half of all optimal solutions to the dCOP and dCOPTW the maximum tour length constraint is active and in the other half the capacity constraint is active.

Time Window Distribution: We consider $m = 2$ types of time windows given by $[0, \frac{1}{2}L]$ and $[\frac{1}{2}L, L]$ with L being the maximum tour length. Which time window is assigned to a location is decided by a fair coin flip.

Performance Metrics: So far, we described how we sample instances $J \in \mathcal{I}$ from a distribution of instances $\mathcal{I}$. Next, we detail the metrics we use to assess the performance of different policies on these instances. We consider two type of metrics. For our detailed analyses, we report the perfect-information (PI) gap of a policy π on an instance $J \in \mathcal{I}$ given by

$$\text{gap}(\pi, J) := 1 - \frac{R^\pi(J)}{R^{\text{PI}}(J)}.$$

R^π denotes the objective value achieved by the policy π on instance $J \in \mathcal{I}$ and $R^{\text{PI}}(J)$ denotes the objective value of solving the instance *ex post*, i.e., under perfect information, to optimality. For training results, we report the approximated PI gap as it is computationally infeasible to compute the true PI gap for all training instances. We define the approximated PI gap of a policy π on an instance $J \in \mathcal{I}$ as

$$\overline{\text{gap}}(\pi, J) := 1 - \frac{R^\pi(J)}{\mathbb{E}_{I \sim \mathcal{I}}[R^{\text{PI}}(I)]}.$$

R^π denotes the objective value achieved by the policy π on instance $J \in \mathcal{I}$ and $\mathbb{E}_{I \sim \mathcal{I}}[R^{\text{PI}}(I)]$ denotes the expected *ex post* optimal objective value if we randomly sample an instance I from the space of instances $\mathcal{I}$.

Benchmarks: Our benchmark policies cover all problem classes of approximate dynamic programming except for online lookahead methods. We omit online lookahead methods because they are computationally too costly and because they tend to underperform on stochastic dynamic transportation problems that share components of knapsack and routing problems (Ulmer and Thomas 2020). As benchmark methods, we employ

1. a value function approximation that uses a decision-space decomposition to avoid searching the combinatorial decision space (VFA Decomposition);
2. a policy function approximation that chooses requests based on their value-to-resource margin (PFA);
3. a cost function approximation that reserves resources in each decision state for later decision states (CFA);
4. and a static optimization method, that searches the decision space by solving a MILP maximizing immediate reward but disregarding future rewards (Static Optimization).

We refer to Appendix B for the implementation details of our benchmark methods.

6. Results

We first discuss the results of the training phase in Subsection 6.1 before we analyze the performance of each policy on 1000 evaluation instances in more detail in Subsection 6.2. For detailed information on our implementations and training procedures, please see Appendix C.

6.1. Training Results

We train all policy on 20.000 realizations of each problem instance. Each policy is allowed 5 seconds of computation time per decision point. No requests are accepted if no solution is found. The observed mean approximated PI gap (sliding window of size 100) over the 20.000 training steps is shown in Figure 5. The x-axes denote the training episodes and the y-axes denote the approximated mean optimality gaps. We observe that all policies significantly improve over the course of training. For small instances with $n = 3, K = 5$, all policies show similar performance. However, for larger and, in particular, more complex instances, i.e., dCOP and dCOPTW with $n = 10$ or $K = 15$, it becomes apparent that the VFA (MILP) policy outperforms all other policies. The RL-CFA (O) policy is noticeably worse than all other methods on these instances. Not shown in Figure 5 are the PFA and CFA policies as their parameter fitting scheme does not result in learning curves.

6.2. Evaluation

We evaluate our proposed policies and the benchmark policies on a set of 1000 realizations of the same six problem instances (this set is excluded from the training instances). Again, each policy is allowed 5 seconds of computation time per decision point. The mean optimality gaps to the perfect information bounds are summarized in Table 1.

In Figure 6, we further summarize the results aggregated by the problem type (*top*) and the instance parameters (*bottom*). The columns in Figure 6 correspond to the results for the dKP, the dCOP, and the dCOPTW, respectively. The x-axis denotes the instance parameters and the y-axis shows the mean optimality gap for each policy. We make the following observations.

The cost of neglecting uncertainty. Table 1 and Figure 6 highlight that solving each decision state statically, without consideration of future uncertainty, is not a viable policy. Indeed, the static optimization policy is dominated by all other methods on all instances. No matter how uncertainty is addressed in the solution method, it always improves on the static policy.

When to learn request values. When analyzing the results for the dKP instances, we observe that the RL-CFA (O) method outperforms all other policies. In these instances, requests' value-to-resource-consumption-ratio is independent of previously chosen requests and the RL-CFA (O) is able to learn reasonable request value manipulations. For instances with routing- or time-window constraints requests' value-to-resource-consumption-ratio is dependent on previously chosen requests and the RL-CFA (O) policy proves to be among the worst policy. Practitioners should therefore evaluate if and how requests' long-term impacts are interdependent before employing a RL-CFA (O) policy.

Figure 5 **Approximated mean PI gap over 20,000 training episodes for all methods and all considered problem instances.**

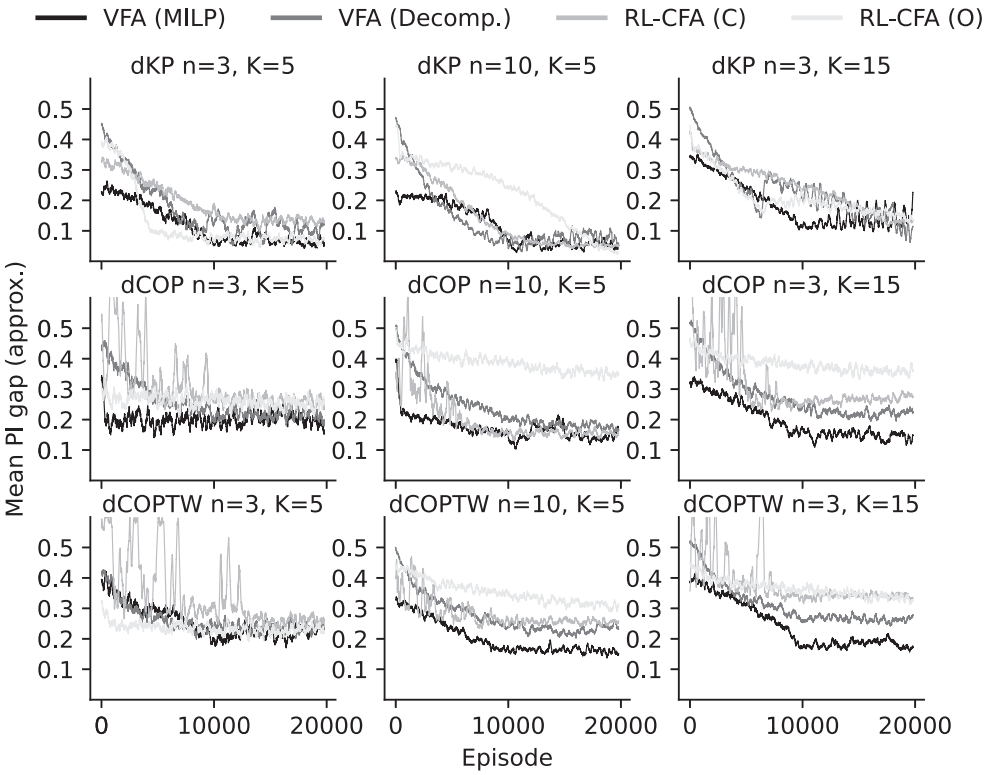
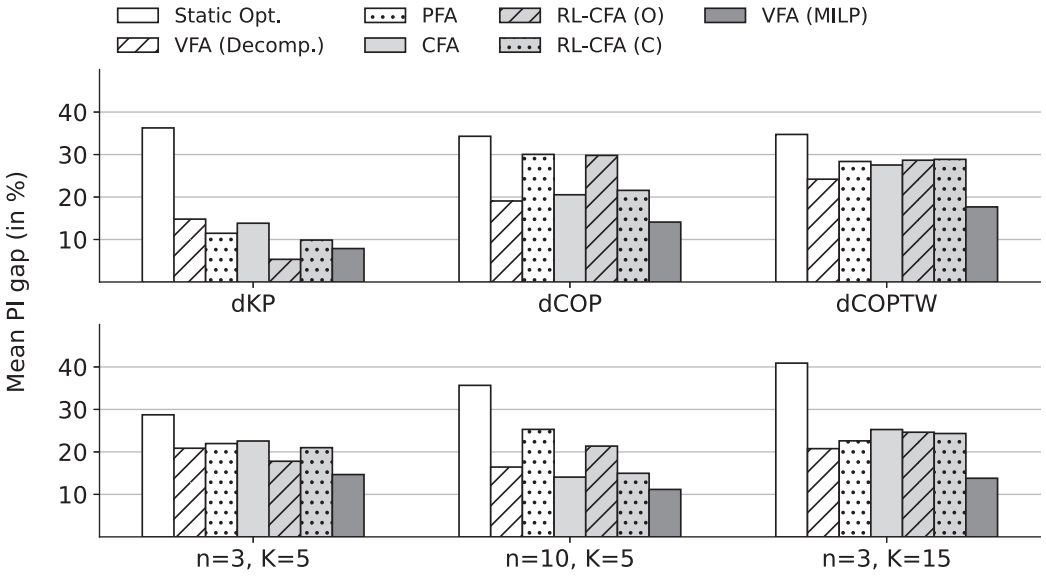


Table 1 **Mean PI gap for 1000 realizations of each instance.**

	dKP			dCOP			dCOPTW			
	n=3 K=5	n=10 K=5	n=3 K=15	n=3 K=5	n=10 K=5	n=3 K=15	n=3 K=5	n=10 K=5	n=3 K=15	Mean
Static Opt.	30.7	36.8	41.3	28.0	34.5	40.4	27.5	35.7	41.0	35.2
VFA (Decomp.)	19.0	9.6	15.8	20.2	17.0	20.0	23.4	22.7	26.5	19.4
CFA	16.0	6.8	18.7	25.6	12.8	23.2	26.1	22.6	33.9	20.6
PFA	15.4	9.6	9.4	26.6	34.2	29.3	23.9	32.1	29.1	23.3
RL-CFA (O)	**6.1**	**3.3**	**6.6**	24.9	31.0	33.5	22.4	29.8	33.8	21.3
RL-CFA (C)	12.9	4.9	11.8	24.7	12.3	27.5	25.4	27.5	33.7	20.1
VFA (MILP)	**6.1**	7.3	10.2	**19.0**	**9.8**	**13.5**	**18.9**	**16.4**	**17.7**	**13.2**

The value of searching and evaluating. For instances with complex constraints (dCOP and dCOPTW), the VFA (MILP) policy performs best. This highlights its capability to handle complex, combinatorial decision spaces. The improvement on the other policies is particularly large for instances with long horizons, i.e, instances with $n = 3, K = 15$. In these instances, the immediate-reward term in the Bellman Equation is dominated by the reward-to-go term. For instances with short horizons and large decision spaces, i.e., instances with $n = 10, K = 5$, a CFA policy that searches the decision space

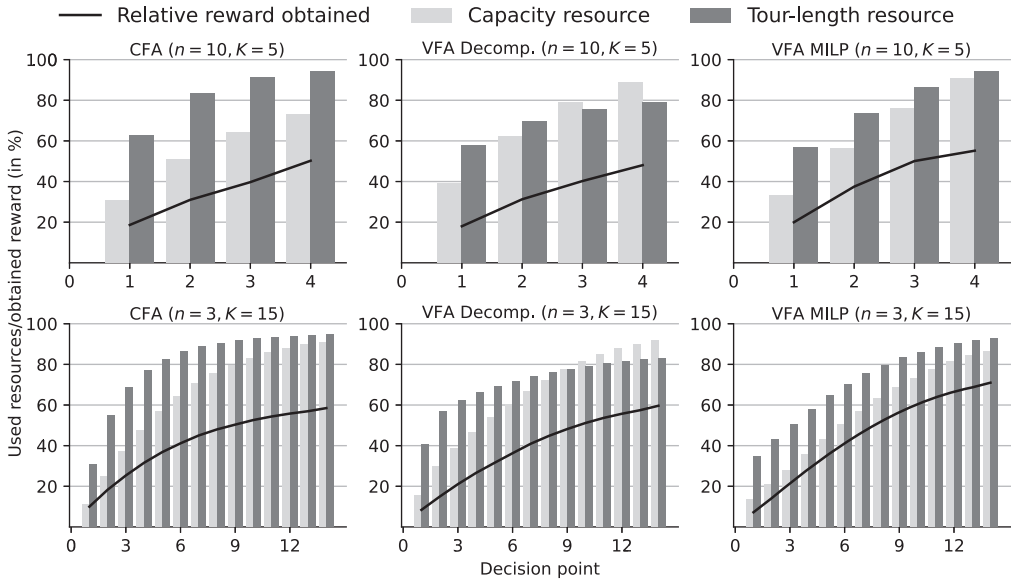
Figure 6 **Observed mean PI gaps for all instances aggregated by problem (*top*) and instance parameters (*bottom*).**



but does not directly evaluate future uncertainty is a viable alternative. As a final remark, the VFA (MILP) policy improves upon the VFA (Decomposition) policy for all considered instances. This is to be expected, as the VFA (MILP) is an extension of the VFA (Decomposition).

When to search and when to evaluate. When a more complex policy, such as the VFA (MILP) policy, is not feasible, we must decide between focusing on searching the decision space and focusing on evaluating decisions with regard to future uncertainty. Our findings, as summarized in Figure 6 (*bottom*), indicate that searching (represented by the CFA policy) is more valuable than evaluating (represented by the VFA Decomposition policy), when the decision space is large and the horizon is short ($n >> K$). In turn, evaluating is more important than searching when the horizon is long and decision space is small ($K >> n$). We further analyze this effect at the example of the dCOPTW. This result highlights that CFAs are viable alternatives to VFAs with decision-space decomposition, which are currently prevalent in the dynamic transportation literature. Figure 7 summarizes how the CFA, VFA (Decomposition), and VFA (MILP) policies behave for dCOPTW instances with either large decision spaces or high uncertainty. The x-axes describe the decision points. The y-axis describes the mean relative expended tour length or capacity as well as the mean relative reward obtained (percentage of the optimal reward) aggregated over 1000 instance realizations. We observe that policies that search the decision space (CFA, VFA MILP) tend to use the tour length resource to its full potential, reaching a tour length of 90% of the allowed tour length in both instances. The VFA (Decomposition) policy is only able to use around 80% of the tour length and, therefore, performs worse than the CFA, when the horizon is short ($n = 10, K = 5$). In turn, the CFA tends to deplete available resources earlier in

Figure 7 **Usage of resources and obtained reward for the CFA, VFA (Decomposition), and VFA (MILP) at the example of the dCOPTW with $n = 10, K = 5$ (*top*) and $n = 3, K = 15$ (*bottom*).**



the decision horizon. For $K = 15$ decision points, the CFA has depleted almost all resources after 8 decision steps, which highlights its limitation to anticipate long horizons. The VFA (Decomposition) policy in comparison, steadily uses the resources over all 15 decision points and yields a higher total reward.

In summary, we observe that reinforcement learning methods are valuable when solving stochastic dynamic transportation problems, but their effectiveness depends on various factors, such as the decision space complexity, the problem horizon, and the instance dimensions.

7. Outlook & Conclusion

Last-mile logistic problems are often characterized by complex, combinatorial decision spaces as well as dynamic decision making due to future uncertainty. Yet, past solution methods tend to focus either on searching the complex, combinatorial decision space or the evaluation of decisions with regard to future uncertainty. This work provided theoretical and empirical arguments in favor of simultaneously searching and evaluating the decision space as well as concrete ideas on how such integrated solutions may be achieved via reinforcement learning. In one approach, a neural network value function is integrated into a mixed integer linear program to solve for decisions that maximize current reward plus expected future reward. In another approach, cost function approximation parameterizations are learned on a state-to-state basis by a policy gradient method.

We see several avenues for future research leading from our proposed solution methods. We highlighted that a cost function approximation parameterized based on state information can improve on a globally

parameterized cost function approximation. Thus, future work may analyze how existing CFAs might be improved via reinforcement learning. For example, in the ride-hailing problem proposed by Riley, Van Hentenryck, and Yuan (2020), postponement of customer services is discouraged by artificial penalty terms. However, in some states, postponement might even be valuable to keep the fleet flexible for future demand. Thus, state-dependent penalties of the CFA (O) might prove valuable. Furthermore, a CFA (C) might be developed to prohibit imbalanced distributions of vehicles in the city. As another example, in Ulmer et al. (2021), artificial penalty terms are integrated to hedge against late deliveries. Here, the proposed concepts might make the penalties state-dependent, e.g., based on the current and expected workload of the fleet. In general, there are many globally parameterized approximate dynamic programming methods that may be extended in a similar way. Obvious examples are policy function approximations, but also lookahead methods have global parameters such as the depth, width, and level of detail of the lookahead procedure. Reinforcement learning provides many suitable methods to learn such a state-dependent parameterization and may be a natural extension of these established methods.

Our findings further indicate that searching the decision space and evaluating decisions with regard to future uncertainty in an integrated approach yields a powerful decision policy. One drawback of integrating a VFA directly into the search is the added complexity in the mixed-integer linear program. Future research may investigate techniques that allow for and improve the integration of more complex and expressive neural networks into mixed-integer linear programs. If solving mixed-integer linear programs with standard solvers is too time-consuming, future research may integrate VFAs into metaheuristics and matheuristics where ideally reinforcement learning does not only evaluate the decisions but also guides the search.

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, project 444657906. We gratefully acknowledge their support. Furthermore, the authors acknowledge the North-German Supercomputing Alliance (HLRN) for providing HPC resources that have contributed to the research results reported in this paper.

References

Agatz N, Campbell A, Fleischmann M, Savelsbergh M, 2011 *Time slot management in attended home delivery. Transportation Science* 45(3):435–449.

Agussurja L, Cheng SF, Lau HC, 2019 *A state aggregation approach for stochastic multiperiod last-mile ride-sharing problems. Transportation Science* 53(1):148–166.

Akkerman F, Mes M, van Jaarsveld W, 2022 *A comparative analysis of neural networks in anticipatory transportation planning*. Preprint at University of Twente.

Anderson R, Huchette J, Ma W, Tjandraatmadja C, Vielma JP, 2020 *Strong mixed-integer programming formulations for trained neural networks. Mathematical Programming* 183(1-2):3–39.

Basso R, Kulcsár B, Sanchez-Diaz I, Qu X, 2022 *Dynamic stochastic electric vehicle routing with safe reinforcement learning. Transportation Research Part E: Logistics and Transportation Review* 157:102496.

Bent RW, Van Hentenryck P, 2004 *Scenario-based planning for partially dynamic vehicle routing with stochastic customers. Operations Research* 52(6):977–987.

Benyahia I, Potvin JY, 1998 *Decision support for vehicle dispatching using genetic programming. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 28(3):306–314.

Bergstra J, Bardenet R, Bengio Y, Kégl B, 2011 *Algorithms for hyper-parameter optimization. Advances in neural information processing systems* 24.

Biggs M, Perakis G, 2020 *Dynamic routing with tree based value function approximations. Available at SSRN 3680162* .

Branchini RM, Armentano VA, Løkketangen A, 2009 *Adaptive granular local search heuristic for a dynamic vehicle routing problem. Computers & Operations Research* 36(11):2955–2968.

Campbell AM, Savelsbergh M, 2006 *Incentive schemes for attended home delivery services. Transportation Science* 40(3):327–341.

Chen X, Hewitt M, Thomas BW, 2018 *An approximate dynamic programming method for the multi-period technician scheduling problem with experience-based service times and stochastic customers. International Journal of Production Economics* 196:122–134.

Chen X, Ulmer MW, Thomas BW, 2022 *Deep q-learning for same-day delivery with vehicles and drones. European Journal of Operational Research* 298(3):939–952.

Chen X, Wang T, Thomas BW, Ulmer MW, 2023 *Same-day delivery with fair customer service. European Journal of Operational Research* 308(2):738–751.

Chen ZL, Xu H, 2006 *Dynamic column generation for dynamic vehicle routing with time windows. Transportation Science* 40(1):74–88.

Chu PC, Beasley JE, 1998 *A genetic algorithm for the multidimensional knapsack problem. Journal of Heuristics* 4(1):63–86.

Dayarian I, Savelsbergh M, Clarke JP, 2020 *Same-day delivery with drone resupply. Transportation Science* 54(1):229–249.

Delarue A, Anderson R, Tjandraatmadja C, 2020 *Reinforcement learning with combinatorial actions: An application to vehicle routing. Advances in Neural Information Processing Systems* 33:609–620.

Fan J, Wang Z, Xie Y, Yang Z, 2020 *A theoretical analysis of deep q-learning*. Bayen AM, Jadbabaie A, Pappas G, Parrilo PA, Recht B, Tomlin C, Zeilinger M, eds., *Proceedings of the 2nd Conference on Learning for Dynamics and Control*, volume 120 of *Proceedings of Machine Learning Research*, 486–489 (PMLR).

Fleckenstein D, Klein R, Steinhardt C, 2023 *Recent advances in integrating demand management and vehicle routing: A methodological review. European Journal of Operational Research* 306(2):499–518.

Gendreau M, Guertin F, Potvin JY, Taillard É, 1999 *Parallel tabu search for real-time vehicle routing and dispatching. Transportation Science* 33(4):381–390.

Goodson JC, Thomas BW, Ohlmann JW, 2017 *A rollout algorithm framework for heuristic solutions to finite-horizon stochastic dynamic programs. European Journal of Operational Research* 258(1):216–229.

Haferkamp J, Ulmer MW, Ehmke JF, 2023 *Heatmap-based decision support for repositioning in ride-sharing systems. Transportation Science* URL `http://dx.doi.org/https://doi.org/10.1287/trsc.2023.1202`.

Heinold A, Meisel F, Ulmer MW, 2022 *Primal-dual value function approximation for stochastic dynamic intermodal transportation with eco-labels. Transportation Science* URL `http://dx.doi.org/https://doi.org/10.1287/trsc.2022.1164`.

Heitmann RJO, Soeffker N, Ulmer MW, Mattfeld DC, 2023 *Combining value function approximation and multiple scenario approach for the effective management of ride-hailing services. EURO Journal on Transportation and Logistics* 12:100104.

Hildebrandt FD, Thomas BW, Ulmer MW, 2023 *Opportunities for reinforcement learning in stochastic dynamic vehicle routing. Computers & Operations Research* 150:106071.

Ichoua S, Gendreau M, Potvin JY, 2000 *Diversion issues in real-time vehicle dispatching. Transportation Science* 34(4):426–438.

Kullman ND, Cousineau M, Goodson JC, Mendoza JE, 2022 *Dynamic ride-hailing with electric vehicles. Transportation Science* 56(3):775–794.

Lei C, Jiang Z, Ouyang Y, 2019 *Path-based dynamic pricing for vehicle allocation in ridesharing systems with fully compliant drivers. Transportation Research Procedia* 38:77–97.

Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, Wierstra D, 2015 *Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971* .

Ma Y, Hao X, Hao J, Lu J, Liu X, Xialiang T, Yuan M, Li Z, Tang J, Meng Z, 2021 *A hierarchical reinforcement learning based optimization framework for large-scale dynamic pickup and delivery problems. Advances in Neural Information Processing Systems* 34:23609–23620.

Miller J, Nie Y, Liu X, 2020 *Hyperpath truck routing in an online freight exchange platform. Transportation Science* 54(6):1676–1696.

Mitrović-Minić S, Laporte G, 2004 *Waiting strategies for the dynamic pickup and delivery problem with time windows. Transportation Research Part B: Methodological* 38(7):635–655.

Papalexopoulos TP, Tjandraatmadja C, Anderson R, Vielma JP, Belanger D, 2022 *Constrained discrete black-box optimization using mixed-integer programming*. Chaudhuri K, Jegelka S, Song L, Szepesvari C, Niu G, Sabato S, eds., *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, 17295–17322 (PMLR).

Powell WB, 2022 *Reinforcement Learning and Stochastic Optimization* (Hoboken, NJ: John Wiley & Sons).

Pureza V, Laporte G, 2008 *Waiting and buffering strategies for the dynamic pickup and delivery problem with time windows. INFOR: Information Systems and Operational Research* 46(3):165–175.

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 AEP, Mes MR, 2017 *Anticipatory freight selection in intermodal long-haul round-trips. Transportation Research Part E: Logistics and Transportation Review* 105:176–194.

Secomandi N, 2001 *A rollout policy for the vehicle routing problem with stochastic demands. Operations Research* 49(5):796–802.

Silva M, Pedroso JP, Viana A, 2023 *Deep reinforcement learning for stochastic last-mile delivery with crowdshipping. EURO Journal on Transportation and Logistics* 12:100105.

Soeffker N, Ulmer MW, Mattfeld DC, 2019 *Adaptive state space partitioning for dynamic decision processes. Business & Information Systems Engineering* 61:261–275.

Soeffker N, Ulmer MW, Mattfeld DC, 2022 *Stochastic dynamic vehicle routing in the light of prescriptive analytics: A review. European Journal of Operational Research* 298(3):801–820.

Sutton RS, Barto AG, 2018 *Reinforcement learning: An introduction* (MIT press).

Thomas BW, 2007 *Waiting strategies for anticipating service requests from known customer locations. Transportation Science* 41(3):319–331.

Ulmer MW, 2020 *Dynamic pricing and routing for same-day delivery. Transportation Science* 54(4):1016–1033.

Ulmer MW, Goodson JC, Mattfeld DC, Hennig M, 2019 *Offline–online approximate dynamic programming for dynamic vehicle routing with stochastic requests. Transportation Science* 53(1):185–202.

Ulmer MW, Mattfeld DC, Köster F, 2018 *Budgeting time for dynamic vehicle routing with stochastic customer requests. Transportation Science* 52(1):20–37.

Ulmer MW, Nowak M, Mattfeld D, Kaminski B, 2020 *Binary driver-customer familiarity in service routing. European Journal of Operational Research* 286(2):477–493.

Ulmer MW, Soeffker N, Mattfeld DC, 2018 *Value function approximation for dynamic multi-period vehicle routing. European Journal of Operational Research* 269(3):883–899.

Ulmer MW, Thomas BW, 2018 *Same-day delivery with heterogeneous fleets of drones and vehicles. Networks* 72(4):475–505.

Ulmer MW, Thomas BW, 2020 *Meso-parametric value function approximation for dynamic customer acceptances in delivery routing. European Journal of Operational Research* 285(1):183–195.

Ulmer MW, Thomas BW, Campbell AM, Woyak N, 2021 *The restaurant meal delivery problem: Dynamic pickup and delivery with deadlines and random ready times. Transportation Science* 55(1):75–100.

Voccia SA, Campbell AM, Thomas BW, 2019 *The same-day delivery problem for online purchases. Transportation Science* 53(1):167–184.

Waßmuth K, Köhler C, Agatz N, Fleischmann M, 2023 *Demand management for attended home delivery–a literature review. European Journal of Operational Research* URL http://dx.doi.org/https://doi.org/10.1016/j.ejor.2023.01.056.

Xie J, Liu Y, Chen N, 2023 *Two-sided deep reinforcement learning for dynamic mobility-on-demand management with mixed autonomy. Transportation Science* URL http://dx.doi.org/https://doi.org/10.1287/trsc.2022.1188.

Yang J, Jaillet P, Mahmassani H, 2004 *Real-time multivehicle truckload pickup and delivery problems. Transportation Science* 38(2):135–148.

Yang X, Strauss AK, 2017 *An approximate dynamic programming approach to attended home delivery management. European Journal of Operational Research* 263(3):935–945.

Zehtabian S, Larsen C, Wøhlk S, 2022 *Estimation of the arrival time of deliveries by occasional drivers in a crowdshipping setting. European Journal of Operational Research* 303(2):616–632.

Appendix

In the Appendix, we will first provide the proofs of our propositions. We will then present the details of the benchmark policies. Finally, we will discuss implementation and training details.

A. Proofs

In this section, we formally prove Proposition (1) and Proposition (2).

Proof of Proposition (1) To prove $\Pi_C = \Pi_V = \Pi_D$, we show that that $\Pi_C \subset \Pi_D, \Pi_V \subset \Pi_D$ as well as $\Pi_D \subset \Pi_C, \Pi_D \subset \Pi_V$.

$\mathbf{\Pi_C \subset \Pi_D, \Pi_V \subset \Pi_D}$: By definition, we have $\Pi_C, \Pi_V \subset \Pi_D$ as these policy classes are deterministic in nature.[1]

$\mathbf{\Pi_D \subset \Pi_V}$: Let $\pi \in \Pi_D$ be an arbitrary deterministic policy. We define π' as

$$\pi'(S_k) = \underset{x_k \in \mathcal{X}(S_k)}{\arg\max}\, p_k^\intercal x_k + \hat{V}(S_k^x), \quad \forall S_k \in \mathcal{S}$$

with

$$\hat{V}(S_k^x) = \begin{cases} M, & x = \pi(S_k) \\ 0, & \text{else.} \end{cases}$$

π' is bounded, piecewise affine and, therefore, we have $\pi' \in \Pi_V$. By choosing $M \in \mathbb{R}$ sufficiently large, the maximum is only attained at $x = \pi(S_k)$ as R_k is bounded by Assumption (1). As there is a unique maximizer, π' is well-defined. By construction, we have $\pi(S_k) = \pi'(S_k), \forall S_k \in \mathcal{S}$ and it follows $\pi \in \Pi_V$, and $\Pi_D \subset \Pi_V$.

$\mathbf{\Pi_D \subset \Pi_C}$: Let $\pi \in \Pi_D$ be an arbitrary deterministic policy. We define π' as

$$\pi'(S_k) = \underset{x_k \in \pi^{\text{MOD-C}}(S_k)}{\arg\max}\, p_k^\intercal x_k, \quad \forall S_k \in \mathcal{S}$$

with $\pi^{\text{MOD-C}}(S_k) = \{\pi(S_k)\} \subset \mathcal{X}(S_k)$. By construction, we have $\pi' \in \Pi_C$. Further, π' is well-defined as there is a unique maximizer. Also by construction, we guarantee $\pi(S_k) = \pi'(S_k)$, $\forall S_k \in \mathcal{S}$ and, therefore, $\pi \in \Pi_C$. Thus, we yield $\Pi_D \subset \Pi_C$.

Next, we consider the case that $x \in \{0,1\}^n, \forall x \in \mathcal{X}$ and show that $\Pi_D = \Pi_O$. Showing $\Pi_O \subset \Pi_D$ follows by definition of Π_V as we only consider deterministic policies. It remains to be shown that $\Pi_D \subset \Pi_O$. Let $\pi \in \Pi_D$ be an arbitrary deterministic policy. We define π' as

$$\pi'(S_k) = \underset{x_k \in \mathcal{X}(S_k)}{\arg\max}\, \pi^{\text{MOD-O}}(S_k)^\intercal x_k, \quad \forall S_k \in \mathcal{S}$$

such that $\pi^{\text{MOD-O}}(S_k) = p_k' \in \mathbb{R}^n$ with

$$p_{ki}' = \begin{cases} 1, & x_{ki} = 1, x_k = \pi(S_k) \\ -1, & else. \end{cases}, \quad i = 1, \dots, n. \tag{29}$$

Again, this yields a unique maximizer given by $x_k = \pi(S_k)$. Thus, π' is well-defined. By construction, we have $\pi' \in \Pi_O$ and $\pi'(S_k) = \pi(S_k), \quad \forall S_k \in \mathcal{S}$. Therefore, we yield $\pi \in \Pi_O$ and $\Pi_D \subset \Pi_O$.

Proof of Proposition (2) To show that $\bar{\pi} := \lim_{i \to \infty} \pi_i$ is optimal, i.e., $\bar{\pi} \in \Pi^*$, we show that $V^{\bar{\pi}}(S) = V(S)$ for all $S \in \mathcal{S}$. As $\{\hat{V}_i\}_{i \in \mathbb{N}}$ converges uniformly to the optimal value function V, we have for every $\delta > 0$, there exists $N \in \mathbb{N}$ such that for all $S \in \mathcal{S}$

$$|V(S) - \hat{V}_n(S)| < \delta, \quad \forall n \geq N. \tag{30}$$

[1] All three policy classes can be extended to allow for stochastic policies. However, this is not the case in our work.

For all terminal states $S_K \in \mathcal{S}$, we have

$$|V^{\pi_n}(S_K) - V(S_K)| = |\max_{x \in \mathcal{X}(S_K)} R(S_K, x) + \hat{V}_n(S_K^x) - \max_{x \in \mathcal{X}(S_K)} R(S_K, x)| < |\max_{x \in \mathcal{X}(S_K)} R(S_K, x) + \delta - \max_{x \in \mathcal{X}(S_K)} R(S_K, x)| = \delta, \quad \forall n > N. \tag{31}$$

Under Assumption (1) of a finite number of decision points, we can apply this argument recursively to get

$$|V^{\pi_n}(S) - V(S)| < K \cdot \delta, \quad \forall n > N. \tag{32}$$

As $K \in \mathbb{N}$ is constant and δ can become arbitrarily small, we get $\lim_{n\to\infty} V^{\pi_n}(S) = V^{\bar{\pi}}(S) = V(S)$ for all $S \in \mathcal{S}$. Therefore, we have $\bar{\pi} \in \Pi^*$.

B. Benchmark Policies

In this part of the appendix, we detail our benchmark policies.

B.1. VFA with a Decision-Space Decomposition

Directly deriving decisions based on a value function approximation for our given sequential decision problems is challenging. We cannot evaluate all possible decisions by enumeration as the decision space features complex constraints and a combinatorial number of decisions. Decomposing an n-dimensional decision x_k into a sequence of n binary decisions $x_k^{(1)}, \dots, x_k^{(n)} \in \{0, 1\}$ is one way to surmount this challenge. In such a decision-space decomposition we sequentially decide for each request presented in a decision point whether to choose it or not. However, it is still not elementary to check if a binary decision is feasible. In the dCOP and dCOPTW, feasibility can only be guaranteed by constructing a tour that visits all chosen requests and fulfills the tour-length and time-window constraints. Therefore, we still must solve a MILP or construct a tour by a dedicated heuristic to ensure feasibility of the binary decision. After each binary sub-decision $x_k^{(i)} \in \{0, 1\}$, the sub-decision state $S_k^{(i)}$ is updated by removing the request that was decided on, updating the state information, and proceeding with the next sub-decision $x_k^{(i+1)}$ until no requests remain. Finally, the sequence of sub-decisions yields a feasible decision $x_k = (x_k^{(1)}, \dots x_k^{(n)}) \in \mathcal{X}(S_k)$ for the actual decision point S_k.

The binary decisions can be made by any feasible reinforcement learning method. We decide for a value function approximation due to its long history in stochastic dynamic transportation problems. We take a request if the reward of the request plus the estimated reward-to-go (estimated by the VFA) of the post-decision state when taking the request is larger than the estimated reward-to-go of the post-decision state when not taking the request. As we are revealed multiple requests in each decision step, we first evaluate all the requests, before choosing the request with the highest reward plus reward-to-go. We repeat this process until it is more profitable to not take any remaining request or no requests remain.

We highlight the VFA with decision-space decomposition at the example of the dKP. Let $\hat{V}$ denote a value function approximation that maps a post-decision state, given by the current time t_k and the current knapsack capacity b_k, to its estimated reward-to-go $\hat{R} \in \mathbb{R}_+$. Then, we can solve a decision state S_k as given in Algorithm 1. The algorithm does not terminate as long as there is an request i with $p_{ki} + \hat{V}_\omega(t_k, b_k - w_{ki}) \geq \hat{V}(t_k, b_k)$. In each iteration, the algorithm loops over all remaining requests. Requests that are infeasible are assigned a value of negative infinity. Otherwise, we save the request's marginal value $\delta_{ki} = p_{ki} + \hat{V}(t_k, b_k - w_{ki}) - \hat{V}(t_k, b_k)$. Once all requests are evaluated, we choose the request $i^* = \arg\max_i \delta_{ki}$ with the highest marginal value if the corresponding marginal value is non-negative.

Algorithm 1 Solving a dKP decision state using a VFA with decision-space decomposition

Require: Value function approximation $\hat{V}$, state $S_k := (t_k, b_k, w_k, p_k)$ with $w_k \in \mathbb{R}^n$, $p_k \in \mathbb{R}^n$.

$x_k \leftarrow [0]_n$
while `True` **do**
  Define $\delta_k \in \mathbb{R}^n$ according to
  $\delta_{ki} = \begin{cases} -\infty, & \text{if } b_k - w_{ki} < 0 \text{ or } x_{ki} = 1 \\ p_{ki} + \hat{V}(t_k, b_k - w_{ki}) - \hat{V}(t_k, b_k), & \text{otherwise} \end{cases}$
  if $\max \delta_k > 0$ **then**
    $x_{ki^*} = 1$ with $i^* = \arg\max \delta_k$
    $b_k \leftarrow b_k - w_{ki^*}$
  else
    return x_k

Otherwise, we terminate the algorithm. If the marginal value is positive, we update $x_{ki^*} = 1$, we update the remaining capacity $b_k \leftarrow b_k - w_{ki^*}$, and remove the request from the list of requests. Algorithm 1 requires a trained VFA $\hat{V}_\omega$. We train the VFA in standard fashion by generating observation-reward tuples during a simulation and saving them in an experience replay. After each simulation we sample random training batches from the experience replay to update the VFA's trainable weights ω. We detail this procedure in the following. Whenever we perform an iteration m in Algorithm 1, we return an updated post-decision state $S_k^{(m)} := (b_k, t_k)$ and the reward $R_k^{(m)} = p_{ki^*}$. Thus, by simulating the entire sequential decision process we obtain a trajectory of length $n \cdot K$ and of the form

$$(S_1^{(1)}, x_1^{(1)}, R_1^{(1)}), \dots, (S_K^{(n)}, x_K^{(n)}, R_K^{(n)}).$$

At the end of one simulation, we calculate the reward-to-go $\bar{R}_k^{(m)} := \sum_{i=k}^{K} \sum_{j=m+1}^{n} R_i^{(j)}$ for each observation $S_k^{(m)}$ and add the tuples to the experience replay. Then, we sample from the experience replay and update the VFA's weights ω according to

$$\omega \leftarrow \omega + \alpha \cdot \nabla_\omega \frac{1}{B} \sum_{i=1}^{B} \left[\hat{V}_\omega(S_i) - R_i \right]^2, \tag{33}$$

where (S_i, R_i), $i = 1, ..., B$ are the sampled experiences.

The observations and rewards-to-go are obtained off-policy, i.e., by following an ϵ-greedy policy. In each decision state, the ϵ-greedy policy either takes a random feasible decision with probability ϵ or by constructing a decision according to Algorithm 1. Further, we choose a linear decay strategy for ϵ

$$\epsilon(h) = \max\left(0, \frac{0.5 \cdot H - h}{0.5 \cdot H}\right).$$

Here, H denotes the number of training simulations to perform. Our VFA (Decomp.) method operates on post-decision states. Therefore, input information is already strongly aggregated and a small network architecture suffices. Thus, we use a feed-forward neural network with ReLU activation and a hidden layer of size 16 for the VFA.

B.2. Policy Function Approximation

Our policy function approximation (PFA) is inspired by Dantzig's famous Knapsack heuristic. It emulates a simple rule-of-thumb on when to accept a request. The PFA iterates over each presented request (sorted by value-to-resource margin) in a state and selects the current request if (i) it is feasible and (ii) its value-to-resource margin is sufficiently high. In the case of the dKP, the decision rule on the value-to-resource margin is defined as

$$p_{ki} - w_{ki} \geq \alpha, \tag{34}$$

where $\alpha \in \mathbb{R}$ is a threshold hyperparameter to be fitted. In the dCOP and dCOPTW, we further require the additional route length δ_{tour} when taking an request to be below a fitted threshold hyperparameter. Thus, we take a request if

$$p_{ki} - w_{ki} \geq \alpha, \tag{35a}$$

$$\delta_{\text{tour}} \quad \leq \beta, \tag{35b}$$

hold true, where $\alpha, \beta \in \mathbb{R}$ are threshold hyperparameters to be fitted. For all instances, the threshold hyperparameters are fitted by a tree-structured Parzen estimator approach optimizing expected improvement Bergstra et al. (2011). We perform 200 hyperparameter search trials of 100 test instances each, amounting to a total of 20,000 training instances.

B.3. Cost Function Approximation

Our cost function approximation (CFA) constraints the knapsack capacity available in each decision step and the maximum tour length that is allowed in each decision step (in the case of the dCOP and dCOPTW). The knapsack capacity b_k is modified according to

$$b'_k = b_k \cdot \alpha \tag{36}$$

with $\alpha \in (0, 1]$ to ensure that the updated capacity is lesser equal than the true capacity but remains positive. In the case of dCOP and dCOPTW, we additionally modify the maximal route length L by

$$L' = L \cdot \beta + L_{k-1} \cdot (1 - \beta) \tag{37}$$

with $\beta \in (0, 1]$ to ensure that the previous route with length L_{k-1} is still feasible. Again, we choose a tree-structured Parzen estimator approach optimizing expected improvement to fit $\alpha, \beta \in (0, 1]$ for all instances. We perform 200 hyperparameter search trials of 100 test instances each, amounting to a total of 20,000 training instances.

B.4. Static Optimization

Treating each decision state isolated, i.e., without consideration of future uncertainty and dynamism, yields a static optimization policy. This policy maximizes the immediate reward in each decision state while ignoring the reward-to-go. The corresponding decision is obtained by solving the combinatorial decision problem given in each decision state to optimality. The static optimization policy serves as a benchmark policy in our experimental study.

C. Implementation and Training Details

In this section, we lay out the implementations details and training procedures of our VFA (MILP) and RL-CFA policies.

Algorithm 2 Fitting the VFA

Require: VFA $\hat{V}_\omega$, learning rate $\alpha(h)$, exploration rate $\epsilon(h)$, total epochs H, batch size B

ER $\leftarrow \{\}$
for $h = 1, \dots, H$ **do**
  $S_1 \sim \mathcal{S}$
  for $k = 1, ..., K$ **do**
    if $z < \epsilon(h)$, $z \sim U(0,1)$ **then**
      $S_k^x, R_k \leftarrow$ `RandomFeasibleDecision`(S_k)
    else
      $S_k^x, R_k \leftarrow$ `Solve Eq. (16)`
    $S_{k+1} \leftarrow S^M(S_k^x)$
  $\bar{R}_k = \sum_{i=k+1}^{K} R_i, \forall k = 1, \dots, K$
  ER $\leftarrow$ ER $\cup \{(S_1^x, \bar{R}_1), \dots, (S_K^x, \bar{R}_K)\}$
  Sample $(S_{\iota(1)}^x, \bar{R}_{\iota(1)}), \dots, (S_{\iota(B)}^x, \bar{R}_{\iota(B)}) \sim$ ER
  Update $\omega \leftarrow \omega + \alpha(h) \cdot \frac{1}{B} \nabla_\omega \sum_{i=1}^{B} [\hat{V}_\omega(S_{\iota(i)}^x) - \bar{R}_{\iota(i)}]^2$
return ω

C.1. VFA (MILP)

The success of integrating a VFA into the objective function of the MILP depends strongly on the prediction quality of the neural network. We train the value function approximation $\hat{V}_\omega$ off-policy by following an ϵ-greedy policy. We summarize how we fit the value function approximation in Algorithm 2. The procedure uses an experience replay (ER) to save tuples of post-decision states and observed rewards-to-go to sample training batches from. The ER is filled by iteratively simulating a random instance of the given sequential decision process starting at a random root state S_1. In each decision state, the ϵ-greedy policy either takes a random feasible decision with probability $\epsilon(n)$, with $\epsilon(n) \to 0$ for $n \to N$, or by solving the MILP extended by the VFA given by Problem (16). After each instance is simulated, we collect the visited post-decision states and observed rewards-to-go, fill our experience replay, and fit the VFA on a random batch of previously observed post-decision-states and rewards-to-go.

Again, as our VFA (MILP) method operates on post-decision states a small network architecture suffices. Thus, we use a feed-forward neural network with ReLU activation and a hidden layer of size 16. We further choose the same linear decay strategy for ϵ

$$\epsilon(s) = \max\left(0, \frac{0.5 \cdot S - s}{0.5 \cdot S}\right)$$

and learning rates are constant ($\alpha = 0.001$).

C.2. Training of the RL-CFA methods

The update procedure for the RL-CFA policies is given in Algorithm 3. The algorithm is summarized as follows:

Algorithm 3 RL-CFA Algorithm

Require: Q_ω, π_θ^{MOD} learning rate $\alpha(h)$,
noise process $\eta(s)$, number of training epochs S, batch size B

ER $\leftarrow$ [ ]
for $h = 1, \dots, H$ **do**
  $S_1 \sim \mathcal{S}$
  for $k = 1, ..., K$ **do**
    $x_k^{\text{MOD}} = \pi_\theta^{\text{MOD}}(S_k) + \eta(h)$
    $S_k^x, R_k \leftarrow \text{MILP}\left(S_k, x_k^{\text{MOD}}\right)$
    $S_{k+1} \leftarrow S^M(S_k^x)$
  for k=1,...,K **do**
    $\bar{R}_k = \sum_{i=k+1}^{K} R_i$
    ER $\leftarrow$ ER $\cup$ $(S_k, x_k^{\text{MOD}}, \bar{R}_k)$
  Sample $(S_{\iota(1)}, x_{\iota(1)}^{\text{MOD}}, \bar{R}_{\iota(1)}), \dots, (S_{\iota(B)}, x_{\iota(B)}^{\text{MOD}}, \bar{R}_{\iota(B)}) \sim$ ER
  Update $\omega \leftarrow \omega + \alpha(h) \cdot \nabla_\omega \sum_{i=1}^{B} [Q_\omega(S_{\iota(i)}, x_{\iota(i)}^{\text{MOD}}) - \bar{R}_{\iota(i)}]^2$
  Update $\theta \leftarrow \theta + \alpha(h) \cdot \frac{1}{B} \sum_{i=1}^{B} \nabla_x Q_\omega(S_{\iota(i)}, x) \nabla_\theta \pi_\theta^{\text{MOD}}(S_{\iota(i)})\big|_{x=\pi_\theta^{\text{MOD}}(S_{\iota(i)})}$
return ω

When simulating an instance, we obtain the trajectory

$$(S_1, x_1^{\text{MOD}}, R_1), (S_2, x_2^{\text{MOD}}, R_2), \dots.$$

We transform the reward to $\bar{R}_i = \sum_{j=i}^{K} R_j$. Again, we save the samples with the modified reward in an experience replay. At the end of each trajectory, we sample B tuples from the experience replay and update Q_ω according to

$$\omega \leftarrow \omega + \alpha \cdot \nabla_\omega \frac{1}{B} \sum_{i=1}^{B} \left[Q_\omega(S_i, x_i^{\text{MOD}}) - \bar{R}_i\right]^2. \tag{38}$$

Then we sample another batch of B experiences and update π_θ^{MOD} according to

$$\theta \leftarrow \theta + \alpha \cdot \frac{1}{B} \sum_{i=1}^{B} \nabla_x Q_\omega(S_i, x) \nabla_\theta \pi_\theta^{\text{MOD}}(S_i)\big|_{x=\pi_\theta^{\text{MOD}}(S_i)}. \tag{39}$$

To ensure exploration during training, we add noise to each decision:

$$x_i^{\text{MOD}} = \pi_\theta^{\text{MOD}}(S_i) + \eta(h). \tag{40}$$

In the case of the dKP, we use Gaussian noise in the form of $\eta(h) \sim \mathcal{N}(0, \sigma(h))$. Here, we choose to decay $\sigma(h)$ linearly according to

$$\sigma(h) = \max\left(0, \frac{0.25 \cdot H - h}{0.25 \cdot H}\right).$$

In the case of the dCOP and dCOPTW, we explore more aggressively (in our preliminary experiments Gaussian noise did not yield good results) and employ Ornstein-Uhlenbeck noise. The noise η is given by

$$\eta(h,t) := x_t \cdot \mu(h),$$

with $\mu(h)$ being a scaling factor

$$\mu(h) := \max\left(0, \frac{0.75 \cdot H - h}{0.75 \cdot H}\right)$$

and x_t being a realization of the Ornstein-Uhlenbeck process defined by

$$x_t = x_0 \exp(-\alpha t) + \gamma(1 - \exp(-\alpha t)) + \beta \exp(-\alpha t) \int_0^t \exp(\alpha p) dW_p,$$

where W_t is some Brownian-motion, γ is the asymptotic mean, α denotes the mean-reversion rate, and β controls the random shocks of the process.

Finally, we choose the network architectures as follows. The RL-CFA methods operate on states, i.e., they are fed high-dimensional information. Thus, we choose a larger feed-forward neural network with ReLU activation and two hidden layers of size 128 for actor and critic of both RL-CFA methods. The actor network when manipulating p has a *tangens hyperbolicus* activation in the output layer and the actor network when manipulating b has a *sigmoid* activation in the output layer to conform with the manipulation decision spaces.

Otto von Guericke University Magdeburg
Faculty of Economics and Management
P.O. Box 4120 | 39016 Magdeburg | Germany

Tel.: +49 (0) 3 91 / 67-1 85 84
Fax: +49 (0) 3 91 / 67-1 21 20

www.fww.ovgu.de/femm

ISSN 1615-4274