A Comparison of Reinforcement Learning Policies for Dynamic Vehicle Routing Problems with Stochastic Customer Requests

Fabian Akkerman[1], Martijn Mes[1], and Willem van Jaarsveld[2]

[1]University of Twente, Enschede, The Netherlands
[2]Eindhoven University of Technology, Eindhoven, The Netherlands

Abstract

This paper presents directions for using reinforcement learning with neural networks for dynamic vehicle routing problems (DVRPs). DVRPs involve sequential decision-making under uncertainty where the expected future consequences are ideally included in current decision-making. A frequently used framework for these problems is approximate dynamic programming (ADP) or reinforcement learning (RL), often in conjunction with a parametric value function approximation (VFA). A straightforward way to use VFA in DVRP is linear regression (LVFA), but more complex, non-linear predictors, e.g., neural network VFAs (NNVFA), are also widely used. Alternatively, we may represent the policy directly, using a linear policy function approximation (LPFA) or neural network PFA (NNPFA). The abundance of policies and design choices complicate the use of neural networks for DVRPs in research and practice. This paper presents an empirical comparison of LVFA, LPFA, NNVFA, and NNPFA policies. The comparison is conducted on several problem variants of the DVRP with stochastic customer requests. To validate our findings, we study realistic extensions of the stylized problem on (i) a same-day parcel pickup and delivery case in the city of Amsterdam, the Netherlands, and (ii) the routing of robots in an automated storage and retrieval system (AS/RS). We find that (i) whether neural network-based approaches or linear policies are better depends subtly on problem characteristics, (ii) the potential ability of neural networks to improve upon linear policies is drawn from their ability to capture complex relationships between state variables and the downstream costs of the state, but (iii) comes at the expense of considerably longer computational times. Furthermore, (iv) supplying engineered features may distort models, but can potentially help the neural network to find a better policy. Finally, (v) in most cases, NNPFAs outperform NNVFAs, while NNVFAs are significantly faster.

Keywords— dynamic vehicle routing, stochastic customer requests, value function approximation, policy function approximation, neural networks, reinforcement learning

1 Introduction

Dynamic Vehicle Routing Problems (DVRPs) are a difficult, relevant, and popular problem in the transportation research field. Through anticipatory decisions in DVRPs, planners can better account for

information that is revealed after decision-making. According to the literature review by Soeffker et al. (2022), DVRPs can be broadly classified into three application areas: *transportation of goods*, e.g., the dynamic pricing and same-day parcel pickup problem where customers' requested delivery moment and location are only known after the pricing and routing decision (Ulmer, 2020), *passenger transport*, e.g., the autonomous ridesharing problem where idle vehicles are hired on demand (Beirigo et al., 2022), and *services*, e.g., the routing of a technician for on-site maintenance (Pham and Kiesmüller, 2022). In this paper, we focus on the DVRP with stochastic customer requests, with as application area the transportation of goods.

The main characteristics of many sequential decision-making problems are: (i) new information is revealed sequentially over multiple periods, (ii) decisions have both a direct effect and a delayed effect on the problem outcomes, and (iii) computing exact solutions for realistically sized problems is intractable due to the well-known "curses of dimensionality". DVRPs involve sequential decision-making under uncertainty and can be formalized as Markov decision processes (MDPs). MDPs of limited size can be solved exactly using stochastic dynamic programming (SDP). For realistically sized instances, approximate methods are necessary. Approximate methods for MDPs are widely used and studied in the transportation research field, see Bouzaiene-Ayari et al. (2016), Ulmer et al. (2019), and Asadi and Nurre Pinkley (2022) for some recent examples. Approximate dynamic programming (ADP) and reinforcement learning (RL) are common approximate frameworks for solving MDPs. ADP/RL enables anticipatory planning by "learning a policy" using a repetitive cycle of decision-making and policy updates during simulation. ADP/RL encompasses a broad range of techniques, and DVRP researchers have devoted more attention to some techniques than to others. Typically, ADP methods learn a value function approximation (VFA), which is an approximate representation of the cost-to-go, i.e., the (estimated) remaining costs until the end of the horizon. Many RL methods also learn a VFA, e.g., Q-learning learns the value of a state-action pair (Sutton and Barto, 2018), but alternatively they directly learn a policy function approximation (PFA), e.g., policy gradient methods like REINFORCE (Williams, 1992). Hybrids between VFA and PFA also exists, e.g., AlphaZero (Silver et al., 2018) and proximal policy optimization (PPO) (Schulman et al., 2017) can to some extent be considered a hybrid.

In general, VFAs and PFAs can be non-parametric, e.g., using a lookup table to store the value and decisions, respectively, or parametric, using a functional representation of the VFA/PFA. Both methods are widely used and have their distinct advantages and disadvantages, as discussed in Bertsimas and Demir (2002), Powell (2011), and Ulmer and Thomas (2020); our focus will be on parametric VFAs and PFAs. For complex and high-dimensional problems, as often found in the transportation research field, parametric VFAs are considered more computationally efficient, but sometimes lack the ability to learn complex relationships (Ulmer and Thomas, 2020). Studies considering parametric VFAs often use a linear approximation (LVFA), mainly combining linear basis functions, which is a form of multiple linear regression (Powell, 2011). Similarly, parametric linear PFAs (LPFAs) often use linear decision rules or linear basis functions to directly approximate a decision. Although computationally efficient and explainable, there are obvious disadvantages to the use of an LVFA/LPFA that might restrict its performance: the cost-to-go of a problem might be too complex to be learned by a linear predictor.

We may use more complex, non-linear, predictors to represent a VFA, e.g., using piecewise linear approximations (Powell, 2011; Ulmer and Thomas, 2020). The alternative use of neural networks (NNVFA) to learn the cost-to-go is studied in many related papers, e.g., van Heeswijk and La Poutré (2019); Zhou et al. (2020); Qin et al. (2020). Another alternative is to use neural networks to *directly* approximate a policy, i.e., neural network policy function approximation (NNPFA); for examples in the transportation domain, see for instance Nazari et al. (2018); Kool et al. (2019); Ma et al. (2021).

The abundance of policies and design choices for neural networks complicate their use in practice and research. These design choices have a large effect on policy performance, but standardized methods for implementing, modelling, selecting, and comparing different decision-making models are hard to find. We do not claim to solve this problem with this paper. However, we attempt to shed light on the possible advantages and disadvantages of NNVFAs or NNPFAs over LVFAs and LPFAs and provide a starting point for future research in this direction. We demonstrate remarkable conceptual similarities between LVFAs, LPFAs, NNVFAs, and NNPFAs. To our knowledge, we present the first empirical comparison of LVFAs, LPFAs, NNVFAs, and NNPFAs for the DVRP with stochastic customer requests. Related comparison studies in the larger transportation field compare neural network-based policies with heuristics, but not with other types of learned policies such as those based on LVFAs or LPFAs, and as such they cannot give insight into the relative advantages of different types of learned policies.

We start our study considering a stylized DVRP, intended to study more fundamental problem characteristics while lacking the fine structure needed in real-world applications. We study the DVRP version with random requests, for which the customers are unknown at the start of the day, and place requests dynamically during the planning horizon (cf. Zhang and Van Woensel, 2023). We consider three different decision type variants for this problem: (i) a basic variant where a policy needs to dynamically decide on single-step vehicle movements, (ii) an assignment problem variant where vehicle routes are constructed using a heuristic and customers need to be dynamically assigned to one of these routes, and (iii) a hybrid problem variant where routes are constructed by a heuristic and customers are assigned to these routes, but the policy is allowed to deviate from the heuristically created routes. In our experiments, we first study stylized problem settings considering all decision variants. Next, we extend variants (ii) and (iii) to two real-world cases. For variant (ii), we study a same-day parcel pickup and delivery case in the city of Amsterdam, the Netherlands. For variant (iii), we study the routing of robots in an automated storage and retrieval system (AS/RS). An overview of the decision space variants and the experimental study is depicted in Figure 1.

Our comparison yields several findings: (i) whether neural network policy based approaches or linear based VFAs are better depends subtly on the problem characteristics, in fact, a slight modification of the problem might result in completely different relative performance of policies, (ii) the potential ability of neural networks to improve upon LVFAs and LPFAs is drawn from their ability to capture the complex relationship between the state and downstream costs of the state, but (iii) comes at the expense of longer computational times. Furthermore, we show that (iv) engineered features can have added value for neural networks, and (v) in most cases, neural network PFAs (NNPFAs) outperform neural network VFAs (NNVFAs). We derive several recommendations for DVRP researchers that consider applying

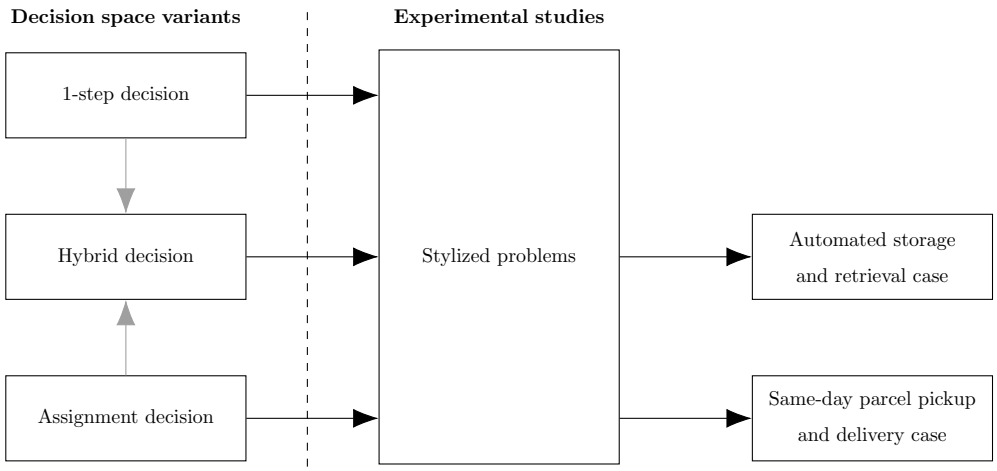


Figure 1: Structure of models and experiments considered in this paper.

learned policies based on these findings.

The remainder of this paper is structured as follows. We review the related literature in Section 2, provide the model in Section 3, and explain the four studied policies, LVFA, LPFA, NNVFA, and NNPFA, in Section 4. We explain the case studies and experimental settings in Section 5, and present our experiments and experimental results in Section 6. Finally, we provide a discussion and conclusion in Section 7.

2 Literature

Our work provides a comparison of parameterized anticipatory policies for DVRPs, with a focus on comparing linear and neural network-based methods. First, we position our paper in the literature on vehicle routing with stochastic customer requests in Section 2.1. Next, we study the use of VFAs and PFAs in the broader transportation domain in Section 2.2, and related work that compares policies. We end with our contribution to the scientific literature.

2.1 Vehicle Routing with Stochastic Customer Requests

The transportation research community increasingly includes stochastic elements in the studied problems, making them intractable for exact methods. We focus on solution methods for the DVRP, which is a VRP where one or more elements include stochasticity (Zhang and Van Woensel, 2023). Although many different sources of uncertainty are considered, the majority of studies consider (i) demand size uncertainty, (ii) request uncertainty, and (iii) environment uncertainty, e.g., travel times (Zhang and Van Woensel, 2023). We focus on request uncertainty, i.e., DVRP studies with stochastic customer requests where customers are added to the vehicle routes during execution. This problem is encountered in practice, for example, for the dispatching of taxi drivers (Beirigo et al., 2022; Liang et al., 2021), dispatching of (autonomous) electric vehicles (Kullman et al., 2022; Dong et al., 2022), express parcel delivery (Ulmer et al., 2018), or the ambulance routing problem (Yi et al., 2010). Beirigo et al. (2022)

consider the stochastic dial-a-ride problem (DARP). They consider autonomous vehicles that can be hired on demand whenever they are idle. They find a policy that decides on the movement of a large fleet by means of a VFA that is trained by an ADP algorithm. In Liang et al. (2021), customers need to be transported from one node to another node by a fleet of taxis. Each customer has a limited time window for waiting for a taxi, which is a stochastic customer attribute. This means that customers who do not want to wait long can only be served by taxis that are close by. For this problem, various RL methods are compared, i.e., advantage actor-critic (A2C), deep Q-networks, and a temporal difference learning method. All RL methods perform better compared to a myopic policy. Furthermore, a centralized dispatching system outperforms decentralized decision-making, where separate agents are modeled as taxi drivers. In Kullman et al. (2022), a fleet of electric vehicles needs to be controlled for a ride-hailing service. They employ deep Q-learning to find a policy for repositioning and recharging decisions. Ulmer et al. (2018) consider an express parcel delivery problem, where decisions consider the acceptance/rejection of customers and the movement of vehicles. They propose an anticipatory method using a dynamic lookup table, where the state is abstracted to the current time and remaining time budget, which is the time that is left until the deadline after serving all currently planned customers. The proposed method shows improvement compared to a myopic method. In Yi et al. (2010), ambulance lateness at accidents is minimized by positioning ambulances in a strategic way, i.e., ambulances that are idle might return to a different base. Soeffker et al. (2022) classify the different solution methods for DVRPs based on the way they anticipate future information. A policy may be *descriptive*, i.e., only acting on the observed state, *predictive*, using information about the exogenous process, e.g., the demand process, and policies can be *prescriptive*, which means that the model uses information about the exogenous process while also considering its interaction with the decision-making policy. In this paper, we focus on prescriptive methods. For a more extensive overview of the state-of-the-art in DVRP, we refer to Ojeda Rios et al. (2021), Soeffker et al. (2022), and Zhang and Van Woensel (2023).

2.2 VFA and PFA Policies in Transportation

According to Powell (2019, 2022), every sequential decision-making problem can be solved using two fundamental strategies. The strategies are (i) policy search, with subclasses (ia) policy function approximation (PFA) and (ib) cost function approximation (CFA), and (ii) lookahead approximation, with subclasses (iia) value function approximation (VFA) and (iib) direct lookahead approximation (DLA). In line with the focus of our study, our review focuses on VFAs and PFAs.

A VFA provides the estimated value of being in a state, which yields a policy that greedily selects a decision with the lowest estimated costs, including both direct and discounted long-term costs. For parametric VFAs, we learn a parameter vector θ to fit a regression model that predicts the downstream costs (Powell, 2011). A widely used parametric VFA in ADP is linear regression, in combination with engineered features, often called the basis functions approach (Powell, 2011). Engineering features often requires a certain level of expertise and problem knowledge, since features need to have a descriptive connection to the state space. Some examples of the successful application of linear regression-based VFAs are given for the intermodal freight selection problem (Pérez Rivera and Mes, 2017), the delivery

dispatch problem (van Heeswijk et al., 2019), and the vehicle routing problem with stochastic demand quantity (Zhang et al., 2022). In transportation research, neural networks are also used as VFA, e.g., applied to the vehicle scheduling problem with stochastic travel times (He et al., 2018), used for learning state-action values for an electric vehicle ride-hailing problem (Kullman et al., 2022), and applied to network traffic signal control (Arel et al., 2010). For a more comprehensive overview of parametric VFAs, we refer to Powell (2011) and Geist and Pietquin (2013). For an extensive review of non-parametric VFA policies, we refer to Bertsekas and Tsitsiklis (1996); Powell (2011) and Ulmer and Thomas (2020). As alternative to VFAs, PFAs provide a direct mapping of states to decisions, i.e., with the state as input they provide the decision as output. PFAs are considered the most widely used policy form, since even simple decision rules could be considered PFAs (Powell, 2022). We focus on the parametric form of PFAs. For parametric PFAs, just like parametric VFAs, we learn a parameter vector θ to fit a model. In contrast to VFAs, PFAs directly parametrize the policy using θ. Linear PFAs are often called affine policies or linear decision rules and are mainly used in the robust optimization research field (El Housni and Goyal, 2021; Powell, 2022). Although linear PFAs provide a tractable solution for many problems, they can be prone to overfitting for low-dimensional problems, and are harder to fit to high-dimensional problems (Powell, 2022). Neural networks as PFA received ample attention in transportation research too, e.g., Nazari et al. (2018); Kool et al. (2019); Mao et al. (2020); Ma et al. (2021), see Hildebrandt et al. (2023) for a comprehensive overview.

Although the application of neural networks in the transportation domain has been explored before, no structured comparison has been done between typical anticipatory methods, like linear VFAs, and neural network based VFAs or PFAs. In fact, only a limited number of papers compare policies from different policy classes. One such comparison of classes is done in Powell and Meisel (2016). Here, PFAs and VFAs are compared on an energy storage problem. From the comparison with different problem variants, Powell and Meisel (2016) conclude that (i) PFA works well when the relationship between state and decision is clear and when policy functions can be easily parameterized, and (ii) VFA works well when downstream costs of a state can easily be approximated. However, it is often unknown upfront whether the relationship between states and decisions are clear and if downstream costs can be easily approximated. In general, parametric VFAs are considered efficient and tractable since they can generalize the value function over many states, but are sometimes inaccurate (Ulmer and Thomas, 2020; Boute et al., 2022). Balaji et al. (2019) compare state-of-the-art RL to baseline methods on three common OR problems: bin packing, the news vendor problem, and the VRP with stochastic customer requests. For the VRP with stochastic customer requests, the baseline is a re-optimization approach employing a MIP, i.e., the baseline disregards anticipation. From their numerical results, they conclude that all RL-policies outperform or at least perform equally well compared to the baseline policy, but they do not provide further guidance for the use and modelling of such RL-policies.

To close the research gap outlined above, we conduct a comparative study of several RL policies on several DVRP variants. Our contribution to the scientific literature is as follows. We provide insights into anticipatory solution methods for DVRPs, and in particular the added value of neural networks. More specifically, we fill the gap in the scientific literature with a study into the advantages, disadvantages,

and performance of neural network policies compared to linear methods. We show results on stylized problems as well as on real-world problems.

3 Problem Formulation

In this section, we provide the model formulation of the studied DVRP. We start with a general introduction of the problem, an overview of notation, and provide the full formulation of the state and decision space, exogenous information, transitions, and the cost function in Section 3.1. Next, we introduce the three variants of the decision space in Section 3.2.

3.1 Model

We consider a grid on which multiple vehicles drive around and serve dynamically appearing customers. Vehicles pick up parcels at customers' home locations and subsequently deliver them to a depot. The goal is to find a central policy that controls all vehicles and serves demand as quickly as possible. Each vehicle has limited capacity, and whenever capacity has been reached, the vehicle has to return to the depot. Decisions are made on a discrete-time, infinite horizon. During each period, all vehicles move to new locations, and possible new customer demand appears. At the end of each period, costs are paid for all customers waiting to be served. At each time step, the vehicles can move to an adjacent cell of the grid. After the move of all vehicles, a possible new customer arrives on a random cell of the grid. An overview of relevant notation is given below.

General variables	
$\mathcal{I}$	the locations on the grid
$l \times l$	the dimension of the grid
$\mathcal{V}$	the fleet of vehicles
K	maximum vehicle capacity
O	depot location
Ω	the set of stochastic realizations
State variables	
$R_{i,v}$	binary that indicates if vehicle v is located at location i
k_v	represents the remaining capacity of vehicle v
D_i	binary that indicates if a customer is located at location i
Policy variables	
a_v	the decision to move vehicle v to an adjacent cell, or stay at the current cell
$B_{i,v}$	if location i is a feasible visiting location for vehicle v
π	the decision-making policy
γ	the discount factor of future costs

We model our problems as discrete, sequential decision-making problems formalized as Markov decision processes (MDPs), described by a state space $\mathcal{S}$, a discrete decision space $\mathcal{A}$, a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and transition dynamics $\mathbb{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$. We represent states $\mathbf{s} \in \mathcal{S}$ and actions $\mathbf{a} \in \mathcal{A}$ by N-and M-dimensional vectors, such that $\mathbf{a} \in \mathbb{N}^N$ and $\mathbf{s} \in \mathbb{N}^M$. A stochastic problem realization, i.e., the appearance of new demand on the grid, is denoted by $w \in \Omega$. A decision-making policy is denoted by $\pi : \mathcal{S} \to \mathcal{A}$ and the state-action value function (the long-term costs of taking an action in a certain state) by $Q^\pi(\mathbf{s}, \mathbf{a}) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r | \mathbf{s}, \pi\right]$, where $\gamma \in [0, 1)$ denotes the discount factor. We aim to

find a policy maximizing the objective function $J = \mathbb{E}_{\boldsymbol{a}\sim\pi}[Q^{\pi}(\mathbf{s}, \mathbf{a})]$.

First, we introduce the remaining relevant problem notation. The grid consists of locations $i \in \mathcal{I}$, for which $|\mathcal{I}| = l \times l$. The attributes of the fleet of vehicles $v \in \mathcal{V}$ are the current location $R_{i,v}$ and the remaining capacity k_v. $R_{i,v}$ indicates if vehicle v is currently located at location i. Furthermore, we use D_i, to indicate if a customer is located at location i. The depot location is indicated by O, and the maximum vehicle capacity by K. We define the possible arrival of a new customer (exogenous information) by w. The state space $\mathcal{S}$ is defined as follows:

$$\begin{aligned}\mathcal{S} = \{(R_{i,v}, k_v, D_i) : \\ \sum_{i\in\mathcal{I}} R_{i,v} = 1, \qquad & \forall v \in \mathcal{V}, && \text{(1a)}\\ 0 \leq k_v \leq K, \qquad & \forall v \in \mathcal{V}, && \text{(1b)}\\ D_i \leq 1, \qquad & \forall i \in \mathcal{I}\}. && \text{(1c)}\end{aligned}$$

Constraint 1a ensures that a vehicle can only be located at a single location on the grid. Constraint 1b ensures that the vehicle capacity is bounded, and Constraint 1c ensures that only a single customer appears on a location on the grid.

The decision space $\mathcal{A}$ is defined by all new locations i of the vehicles. Hence, the decision can be represented by:

$$\mathcal{A} = \{(a_v) : B_{a_v,v} = 1, \quad \forall v \in \mathcal{V}\}. \tag{2a}$$

Here, the set $B_{i,v}$ represents all feasible visiting locations of vehicle v. This ensures that (i) vehicles do not move outside the grid, (ii) vehicles can only move to an adjacent cell, and (iii) that a vehicle takes the shortest path back to the depot whenever its capacity has been reached. Note that it is allowed to preemptively return to the depot.

The transition to a new state involves (i) the potential movement of all vehicles, (ii) the possible serving of customer demand, (iii) the possible delivery of customer parcels at the depot, and (iv) exogenous information $w \in \Omega$. Here, w is the location of a new customer, if no demand appears, w is empty. Accordingly, we define the following transitions of the state variables. We define the transition for a single vehicle $v \in \mathcal{V}$ for brevity, but the transitions apply to the complete fleet. For simplicity, we denote the information related to the previous state by s^{old}. The movement of a vehicle is denoted by:

$$R_{i,v} = \begin{cases} 1, \text{ if } i = a_v(s^{old}), & \forall i \in \mathcal{I}, \\ 0, \text{ otherwise}, & \forall i \in \mathcal{I}. \end{cases} \tag{3}$$

The serving of customer demand is denoted by:

$$D_i = D_i(s^{old}) - R_{i,v}(s^{old}) \implies k_v(s^{old}) > 0, \forall i \in \mathcal{I}, \tag{4}$$

where demand can only be picked up if the respective vehicle has remaining capacity. The vehicle capacity is updated as follows:

$$k_v = k_v(s^{old}) - 1 \implies D_i(s^{old}) = R_{i,v}(s^{old}) = 1, \tag{5}$$

and the delivery of customer parcels at the depot, and the subsequent reset of vehicle capacity is denoted by:

$$k_v = K \implies R_{x,v}(s^{old}) = 1, \text{ where } x = O. \tag{6}$$

Finally, the customer demand is updated with the newly appeared customer:

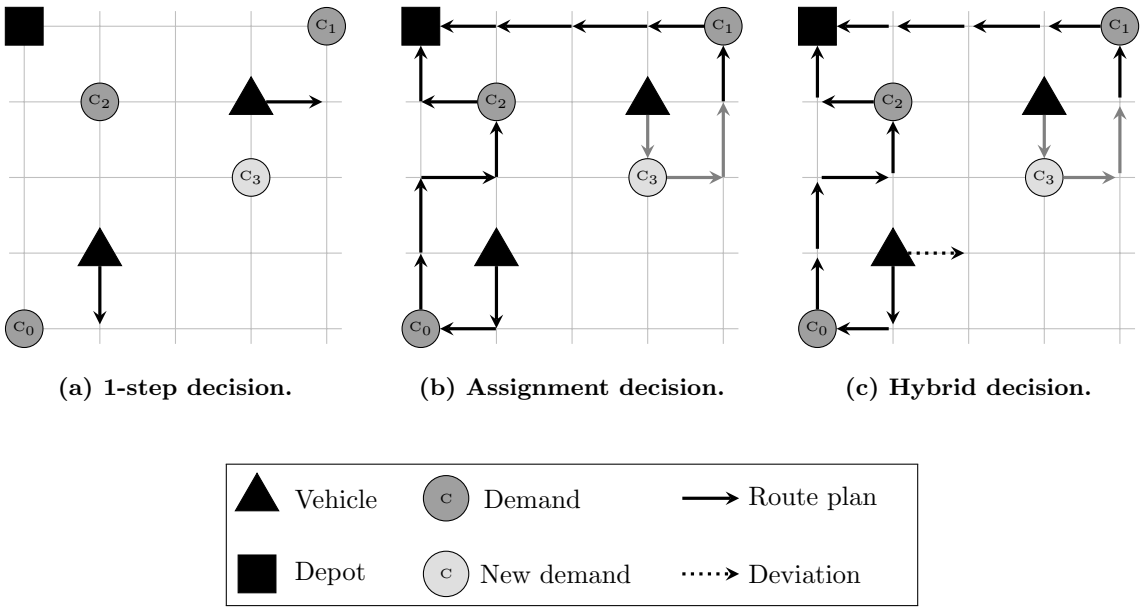
$$D_w = 1. \tag{7}$$

To finalize the formulation, we provide the cost function. A fixed cost of h is paid for every customer waiting for pickup. So, the cost function is the following:

$$r = h \sum_{i \in \mathcal{I}} D_i. \tag{8}$$

3.2 Decision Space Variants

We propose several variants of the decision space as presented in Section 3.1. By altering the decision space, we can test different settings and show how the different policies compare. The three variants of the decision space are illustrated in Figure 2.



(a) 1-step decision. **(b) Assignment decision.** **(c) Hybrid decision.**

Figure 2: Illustration of the decision space variants with a 5×5 grid and 2 vehicles.

Figure 2a shows the standard 1-step decision, as presented in Section 3.1. Here, all vehicles decide on their next location at every period. This problem requires anticipation and cooperation of vehicles to serve customers as quickly as possible, e.g., a policy can anticipate future customers by moving vehicles to a strategic location whenever they are idle. Next, Figure 2b shows a different decision space. Now, the vehicle routes are pre-planned, and all vehicles have to adhere to their route. The routes are determined using an external policy, and are not part of the model. Whenever a new customer appears, the decision-maker has to decide on the assignment of the customer to one of the routes. After every new assignment,

a heuristic finds the shortest route that serves all customers. A policy can anticipate and save costs by assigning customers to vehicles that are expected to come close by, i.e., vehicles need to be spread over the grid. The changes to the model are as follows. First, the state is extended with the set of already accepted customer locations $\mathcal{G}_v$ per vehicle. Next, the decision is altered:

$$\mathcal{A} = \{(a) : 1 \leq a \leq |\mathcal{V}|, \qquad \forall v \in \mathcal{V}\}. \tag{9a}$$

Here, the potential new demand that arrived is assigned to one of the vehicle routes. Constraint 9a ensures that a new customer can only be assigned to an existing vehicle. All other constraints on the decision space are lifted since the external routing policy ensures that only feasible decisions are taken. Depot returns are included in the routing policy such that customers are planned after a depot trip whenever a vehicle has reached its capacity. The routing policy employs cheapest insertion to determine the routes.

Finally, we consider a hybrid of the first two decision spaces, as shown in Figure 2c. Here, vehicles normally follow the assigned route, with the decision space presented in Figure 2b. However, policies can deviate from this route by making a sidestep. After a deviation step, the route serving all assigned customers is recalculated and followed again, unless another sidestep is made. Hence, the decision is comprised of two phases: (i) the assignment of a new customer to a route and (ii) the possible deviation from a route. Note that by means of this deviation decision, demand may be served by a vehicle different from the one that was intended to serve the customer. For brevity, we omit the changes to the model as it is a combination of the previously presented models.

4 Parameterized Policies

The goal of this section is to explain notation and show differences and similarities of the studied policies. Parameterized policies π_θ map states s to decisions $a = \pi_\theta(s)$, where θ represents the policy *parameters*. A learning algorithm is responsible for identifying parameters θ so that the policy adequately considers the direct and delayed effects of the current decision (Powell, 2022). We compare four parameterized policies in two fundamental categories (cf. Powell, 2022): value function approximation (LVFA and NNVFA) and policy function approximation (LPFA and NNPFA).

Every learning-based policy requires a training algorithm. The chosen policy restricts the choice of algorithms. Since algorithmic comparisons are outside the scope of this paper, we stay close to the generic, textbook algorithms, or use a natural extension of them. In general, the training of learning-based policies follows a cyclical process of two phases: (i) data collection and (ii) learning a policy π. During the data collection phase, samples of observed states, decisions, and costs are collected, while using the policy learned in the previous iteration. Next, using the collected data, the learned policy is updated in phase (ii). The policy π is updated by minimizing a loss function to tune the trainable weights θ. This cyclical process continues until the learned policy converges. Although the training algorithms for VFAs and PFAs are different, many similarities exist. For LVFA we use an approximate value iteration (AVI) algorithm (cf. Powell, 2011), for NNVFA we use a natural extension of this algorithm suited for neural networks (batch-AVI), and for both LPFA and NNPFA, we use an extension of the

approximate policy iteration algorithm (batch-API) (cf. van Jaarsveld, 2020). Note that we artificially split the learned policy and the training algorithm for the sake of explainability. In this section, we focus on the learned policies. The explanation of the training algorithms to obtain the policies can be found in the appendix.

In the remainder of this section, we discuss the policies in Section 4.1, while the engineered and state-literal features for the DVRP are discussed in Section 4.2. Finally, we discuss the loss functions used for the policies in Section 4.3.

4.1 VFA and PFA Policies

Remember that s denotes a state for which a decision will be taken, and let $\mathcal{A}(s)$ denote possible decisions. For $a \in \mathcal{A}(s)$, the post-decision state s^a denotes the state of the system immediately after taking decision a in state s. The post-decision state is often used for value function approximation since it avoids the need to calculate the expectations (Powell, 2011). Now we define the four policy types.

LVFA and NNVFA policies make decisions based on approximations of the expected downstream costs of a state s^a. The LVFA approximation is based on engineered features that are denoted by $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_n)$; we will write $\boldsymbol{\phi}(s^a)$ to make explicit that features are calculated over the post-decision state. The LVFA is a linear/affine approximation of downstream costs:

$$\overline{V}_\theta(s^a) = \mathbf{a} \cdot \big(\boldsymbol{\phi}(s^a)\big) + b = a_1\phi_1(s^a) + \ldots + a_n\phi_n(s^a) + b. \tag{10}$$

Here, $\theta = (\mathbf{a}, b)$ are the trainable weights for the LVFA, with $\mathbf{a}$ being an $n \times 1$ vector and b a scalar. The LVFA policy chooses the decision that minimizes the direct costs $C(s, a)$ observed with the transition to the post-decision state, and the expected downstream costs $\overline{V}_\theta(s^a)$, discounted with γ:

$$\pi_\theta^{\text{LVFA}}(s) = \underset{a \in \mathcal{A}}{\arg\min} \left[C(s, a) + \gamma\, \overline{V}_\theta(s^a) \right]. \tag{11}$$

The LPFA directly represents a policy, i.e., it directly provides the decision as output. We use softmax regression as predictor, which is a generalization of logistic regression for multiple output labels (Géron, 2017). Although the softmax function itself is non-linear, the outputs of softmax regression are determined by a linear/affine transformation of the input features, and therefore softmax regression can be considered a linear model, where probabilities of decisions are determined with:

$$\sigma\big(\widehat{p}^a_{\breve{\theta}}(s)\big)^{\text{LPFA}} = \sigma\big(\mathbf{a} \cdot \big(\boldsymbol{\phi}(s)\big) + b = a_1\phi_1(s) + \ldots + a_n\phi_n(s) + b\big), \tag{12}$$

where $\sigma(\widehat{p}^a)$ is the probability assigned to decision a and $\breve{\theta} = (\mathbf{a}, b)$ are the trainable weights. Next, the policy can be denoted by:

$$\pi_{\breve{\theta}}^{\text{LPFA}}(s) = \underset{a \in \mathcal{A}}{\arg\max} \left[\sigma(\widehat{p}_{\breve{\theta}}(s))_a \right]. \tag{13}$$

For NNVFA and NNPFA, we adopt the standard multi-layer perceptron (MLP) as neural network (cf. Gijsbrechts et al., 2022); other options are discussed in Section 7. NNVFA uses a neural network $\mathcal{N}_{\tilde{\theta}}(\cdot) \in \mathbb{R}$ to approximate the downstream costs for s^a. The networks argument is a literal representation of the

post-decision state that will be denoted by $\boldsymbol{\psi}(s^a) = (\psi_1, \ldots, \psi_m)$, yielding the following approximation and policy:

$$\overline{V}^N_{\tilde{\theta}}(s^a) = \mathcal{N}_{\tilde{\theta}}\big(\boldsymbol{\psi}(s^a)\big), \tag{14}$$

$$\pi^{\text{NNVFA}}_{\tilde{\theta}}(s) = \underset{a \in \mathcal{A}}{\arg\min} \left[C(s,a) + \gamma\, \mathcal{N}_{\tilde{\theta}}\big(\boldsymbol{\psi}(s^a)\big) \right]. \tag{15}$$

To better compare (10) and (14), i.e., the policy used by LVFA and NNVFA, suppose $\mathcal{N}_\theta$ has a single hidden layer. Then

$$\mathcal{N}_{\tilde{\theta}}\big(\boldsymbol{\psi}(s^a)\big) = \mathbf{a} \cdot \Big(A_1 \boldsymbol{\psi}(s^a) + \mathbf{b}_1 \Big)^+ + b, \tag{16}$$

where $\tilde{\theta} = (\mathbf{a}, b, A_1, \mathbf{b}_1)$ are the trainable weights for the NNVFA. Note that the hidden layer activation corresponds to the $p \times 1$ vector $(A_1 \boldsymbol{\psi}(s^a) + \mathbf{b}_1)^+$: its role in (16) is similar to the role of $\boldsymbol{\phi}$ in (10). That is, where the LVFA relies on handcrafted features, the NNVFA relies on part of the neural network to compute appropriate features from the state-literal representation $\psi(s^a)$.

While our NNVFA policy employs a neural network with a scalar output that approximates the expected downstream costs, the NNPFA policy adopts a neural network $\mathcal{N}'$ that directly represents the policy. For that purpose, $\mathcal{N}'$ takes as input the state-literal representation $\boldsymbol{\psi}(s)$ of a (pre-decision) state s, and outputs a *vector* in $\mathbb{R}^{|\mathcal{A}|}$. Let $\Big(\mathcal{N}'_{\tilde{\theta}}\big(\boldsymbol{\psi}(s)\big)\Big)_a$ denote the a^{th} element of this vector, i.e., the element that corresponds to decision a. The NNPFA policy is now defined as follows:

$$\pi^{\text{NNPFA}}_{\tilde{\theta}}(s) = \underset{a \in \mathcal{A}}{\arg\max} \left[\Big(\mathcal{N}'_{\tilde{\theta}}\big(\boldsymbol{\psi}(s)\big)\Big)_a \right]. \tag{17}$$

The neural network policies in (15) and (17) are quite similar, but there is a key difference: the neural network in (15) approximates the downstream costs for s^a, which are subsequently used to select a cost-optimal decision, while the neural network in (17) provides a vector of "scores" based on the state s; one for each decision. From (11), (13), (15), and (17), the similarities between the four types of studied policies are remarkable; see Table 1 for an overview.

Table 1: Overview of the focal parameterized policies and corresponding notation.

Policy	Input state	Features	RL approach	ML model
LVFA	post-decision s^a	engineered $\boldsymbol{\phi}(s^a)$	via VFA	linear regression
LPFA	pre-decision s	engineered $\boldsymbol{\phi}(s)$	direct PFA	softmax regression
NNVFA	post-decision s^a	state-literal $\boldsymbol{\psi}(s^a)$	via VFA	neural network
NNPFA	pre-decision s	state-literal $\boldsymbol{\psi}(s)$	direct PFA	neural network

4.2 Features for the Stochastic Dynamic VRP

The common way of providing state information to linear policies is by means of handcrafted features, also called the basis function approach. Providing handcrafted features is often necessary since the linear policies do not have the capability to infer a feature representation from the state space. Deep learning approaches are capable of directly interpreting the state space and do not require handcrafted features.

To make the comparison of policies somewhat fair, we conduct separate experiments where all policies get the same features as input. For all other experiments, we provide engineered features to the linear policies and only the state to the neural network policies. In this section, we first provide the handcrafted features, denoted by ϕ, after which we provide the features as provided to the neural network, denoted by ψ.

Based on numerical experiments with a large set of features ϕ, we found a smaller feature set that provides a high predictive capability for the linear policies. We found that features that describe the *global system* instead of individual vehicles provide the best results. We use a function $d(i, j)$, to denote the distance between location i and j. Next, we define the following basis functions ϕ as used for the linear policies:

- ϕ^1: Number of waiting customers, i.e., $\sum_{i \in \mathcal{I}} D_i$.
- ϕ^2: Cumulative distance between vehicles and demand, i.e., $\sum_{v \in \mathcal{V}} \sum_{i \in \mathcal{I}} d(R_{x,v}, D_i)$, where $R_{x,v} = 1$ and $D_i = 1$.
- ϕ^3: Cumulative distance between vehicles, i.e., $\sum_{v \in \mathcal{V}} \sum_{v' \in \mathcal{V}} d(R_{x,v}, R_{y,v'})$, where $R_{x,v} = 1$ and $R_{y,v'} = 1$.
- ϕ^4: Sum of remaining vehicle capacity, i.e., $\sum_{v \in \mathcal{V}} k_v$.
- ϕ^5: Cumulative distance between vehicles and depot, i.e., $\sum_{v \in \mathcal{V}} d(R_{x,v}, O)$, where $R_{x,v} = 1$.
- ϕ^6: Sum of remaining vehicle capacity $\times$ distance to the depot, i.e., $\sum_{v \in \mathcal{V}} k_v \cdot d(R_{x,v}, O)$.
- ϕ^7: Variance of the number of vehicles in each quadrant, with $\text{Var}(\mathcal{L})$, where $l_q \in \mathcal{L}$ counts the number of vehicles in each quadrant q of the grid, i.e., $l_q = \sum_{v \in \mathcal{V}} \mathbb{1}_{q,v}$, where the indicator function $\mathbb{1}_{q,v}$ is equal to 1 if vehicle v is located in quadrant q.
- ϕ^8: Number of vehicles located on the edge of the grid, i.e., $\sum_{v \in \mathcal{V}} \mathbb{1}_v$, where the indicator function $\mathbb{1}_v$ is equal to 1 if vehicle v is located on the edge of the grid.

For the alternative decision space variants where customers are assigned to a route, the state space is extended with a list of assigned customers per vehicle, denoted by $\mathcal{G}_v$. This additional state information is represented by the following basis functions:

- ϕ^9: Average distance between assigned customers per vehicle, i.e., $\frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} \sum_{g \in \mathcal{G}_v} \sum_{g' \in \mathcal{G}_v} d(g, g')$.
- ϕ^{10}: Variance of distance between assigned customers, i.e., $\text{Var}(\mathcal{M})$, where $m_v \in \mathcal{M}$ is the distance between assigned customer per vehicle v, i.e., $m_v = \sum_{g \in \mathcal{G}_v} \sum_{g' \in \mathcal{G}_v} d(g, g')$.
- ϕ^{11}: Variance of the number of assigned customers per vehicle, i.e., $\text{Var}(\mathcal{P})$, where $p_v \in \mathcal{P}$ counts the number of assigned customers, i.e., $p_v = |\mathcal{G}_v|$.

Aside from the basis functions, we define several features that we consider "state-literals", i.e., not handcrafted but a direct representation of the state. We denote state literals with ψ:

- $\boldsymbol{\psi}^1$: All vehicle locations $R_{i,v}$, i.e., a one-hot encoded vector per vehicle.
- $\boldsymbol{\psi}^2$: All demand locations D_i, i.e., a one-hot encoded vector for the entire grid.
- $\boldsymbol{\psi}^3$: The remaining capacity per vehicle, i.e., k_v.

In case of the assignment decision space, we replace $\boldsymbol{\psi}^2$ by two features:

- $\boldsymbol{\psi}^4$: All assigned but unserved demand locations per vehicle $\mathcal{G}_v$, i.e., a one-hot encoded vector per vehicle indicating the demand locations.
- $\boldsymbol{\psi}^5$: The location of the new and still unassigned demand.

For a more detailed study and comparison of features for describing VRP instances, we refer to Arnold and Sörensen (2019); Nicola et al. (2019); Akkerman and Mes (2022).

4.3 Loss Functions and Parameter Updates

After the data collection phase, the policy π needs to be updated. For each policy, we require a *loss function* in order to tune the trainable weights. Tuning is an iterative process where we update weights θ by moving in the opposite direction of the gradient of the loss.

We tune our LVFA policy using recursive least squares, i.e., after *every* new observation $\widehat{v}$, weights θ are updated to minimize the loss. Our loss minimizes the weighted square error between model predictions and samples, i.e.,

$$\text{loss}_\theta^{\text{LVFA}} = \sum_{i=1}^{m} \lambda_i \Big(\widehat{v}_i - \overline{V}_\theta(s_i^a)\Big)^2 = \sum_{i=1}^{m} \lambda_i \Big(\widehat{v}_i - \mathbf{a} \cdot (\boldsymbol{\phi}(s_i^a)) - b\Big)^2, \tag{18}$$

where $\theta = (\mathbf{a}, b)$, m is the current data set size, and λ_i is the weight associated with sample i. Since we sequentially update our policy, newer observations are typically more reliable and should receive more weight, hence, we adopt recursive least squares for *non-stationary data*, which resembles stochastic gradient descent: the weight λ_i assigned to new observations is iteratively increased as the LVFA weights stabilize (cf. Powell, 2011), and consequently, the importance of older observations decreases.

For LPFA, NNVFA and NNPFA, we instead obtain *batches* of observations, and minimize the loss over the batch while weighing individual observations equally. With this, the NNVFA loss becomes:

$$\text{loss}_{\tilde{\theta}}^{\text{NNVFA}} = \sum_{i=1}^{m} \frac{1}{m} \Big(\widehat{v}_i - \overline{V}_{\tilde{\theta}}^{N}(s_i^a)\Big)^2 = \sum_{i=1}^{m} \frac{1}{m} \Big(\widehat{v}_i - \mathcal{N}_{\tilde{\theta}}(\boldsymbol{\psi}(s_i^a))\Big)^2. \tag{19}$$

Note that here, m denotes the size of the batch that is propagated through the neural network.

The LPFA and NNPFA policies share the same loss function. The loss similarly measures the distance between the model predictions and the information collected in the samples. However, LVFA and NNVFA predict the value corresponding to a post-decision state, whereas the PFAs predict a *decision* corresponding to a pre-decision state, i.e., the PFA policies predict a *categorical* variable. Clearly, the L_2 norm adopted for LVFA and NNVFA is an inappropriate measure of the loss in the categorical case. Instead, the output of the neural network is interpreted as a stochastic policy by applying a soft argmax operator, and the loss for a sample becomes the cross-entropy between i) the probability distribution

over the feasible decisions $a \in \mathcal{A}(s)$ as outputted by the neural network, and ii) the decision $\widehat{a} \in \mathcal{A}(s)$ that resulted in the lowest downstream costs $\mathcal{Z}$, as obtained during sample collection (cf. Silver et al., 2018). So, we minimize:

$$\text{loss}_{\bar{\theta}}^{\text{PFA}} = -\sum_{i=1}^{m} \frac{1}{m} \log \frac{\exp[\left(\mathcal{N}'_{\bar{\theta}}(\boldsymbol{\psi}(s_i))\right)_{\widehat{a}_i}]}{\sum_{a \in \mathcal{A}(s)} \exp[\left(\mathcal{N}'_{\bar{\theta}}(\boldsymbol{\psi}(s_i))\right)_a]}, \tag{20}$$

where s_i and $\widehat{a}_i$ are the state and corresponding decision collected for sample i, and $\exp[\left(\mathcal{N}'_{\bar{\theta}}(\boldsymbol{\psi}(s_i))_{\widehat{a}_i}\right)] / \sum_{a \in \mathcal{A}(s)} \exp[\left(\mathcal{N}'_{\bar{\theta}}(\boldsymbol{\psi}(s_i))\right)_a]$ denotes the probability assigned by the neural network to decision $\widehat{a}_i$ for state s_i. Hence, the loss decreases as the neural network assigns a higher probability to the correct label $\widehat{a}_i$.

For LPFA, NNVFA, and NNPFA, the loss is minimized using adaptive moment estimation (Adam), see Kingma and Ba (2014) for details.

5 Case Studies and Experimental Settings

In this section, we explain our experiments. First, we present the experimental settings for the stylized problem in Section 5.1. Next, we present the same-day parcel pickup and delivery case in Section 5.2 and finally, we explain the AS/RS case in Section 5.3.

5.1 Settings for the Stylized Problem

Note that we define and study a stylized problem for all three decision space variants: the 1-step decision, the assignment decision, and the hybrid decision. For the stylized problem, we consider a 10×10 grid where 5 vehicles operate and serve customers. The depot is located in the top left corner of the grid. Vehicles have a maximum capacity of 5 parcels. The initial locations of vehicles and customer locations are randomly drawn. On average, every 2 time steps a new customer arrives. Vehicles travel in Manhattan style and each time step t they can travel to neighboring locations. We make several assumptions:

1. *Demand and Customers*

   Only a single customer appears per period after the decision-making, and a customer cannot appear in a location where a different customer is already waiting to be served. Serving customers can be postponed indefinitely, i.e., we do not consider time windows for serving customers. The order of serving customers is up to the decision-maker, and no penalties or rewards are related to the individual waiting time per customer. When demand appears at a location where a vehicle is positioned, the demand is directly served, if the vehicle capacity allows it.

2. *Vehicles and Grid*

   The movement of the vehicles is in a Manhattan style, and vehicles can only move one step per period. We consider a homogeneous fleet with equal vehicle speed and capacity. When a vehicle has reached it capacity, it takes the shortest path back to the depot and cannot serve customers until the depot has been reached. After arriving at the depot, the vehicle is directly available for

serving customers again. The movement of vehicles is limited by the edges of the grid. Vehicles can be in the same location at the same time step.

5.2 Same-day Parcel Pickup and Delivery Case

For the assignment decision space variant, we consider a real-world problem of the same-day pickup of customer return parcels in the city of Amsterdam, the Netherlands, as depicted in Figure 3. A third-party logistics service provider takes care of all after-sales logistics for several online large appliance retailers, i.e.., returned or damaged goods need to be picked up at customer locations and delivered to a central depot. The depot location is indicated with blue in Figure 3. The city of Amsterdam is served by a large homogeneous fleet of vehicles with each vehicle having a capacity of 20 parcels. When the capacity has been reached, the vehicle needs to return to the depot before service can continue. During the day, new orders arrive and need to be assigned to a vehicle. We consider the actual road network distances and travel times, including average congestion observed at 2pm. The expected number of orders per day is 300. The spatial distribution of customers is depicted in Figure 3. We draw customer locations relative to the distribution of residential addresses within the different districts of Amsterdam. The operational area is, approximately 81 km^2. Customer arrivals are spread uniformly over the day. Key performance indicators are the total traveled distance, average waiting time per customer, utilized vehicles, and the total costs.

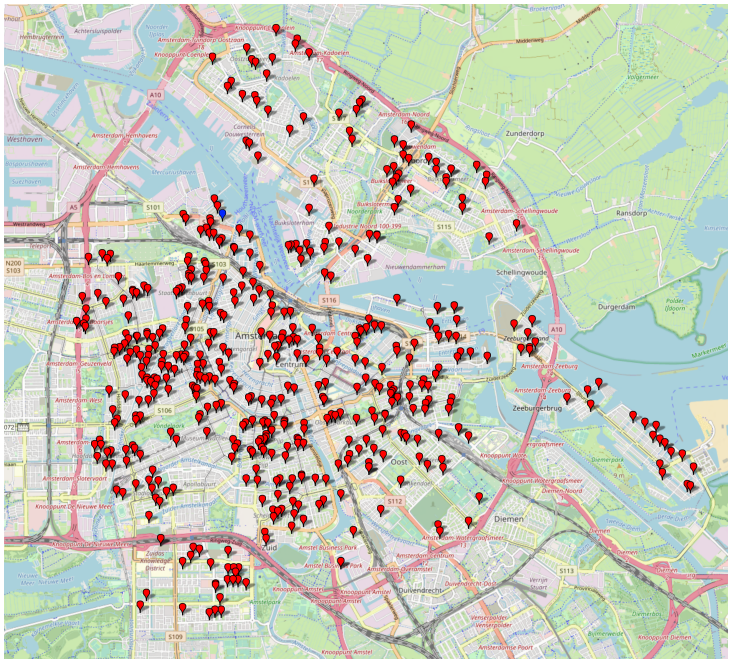


Figure 3: Map of the parcel pickup locations in Amsterdam, the Netherlands (source: OpenStreetMap)

We alter the model as presented in Section 3 by changing the cost function. This way, long customer waiting times are penalized. We define a new state variable Y_i, which is the time that customer i has

already waited. Additionally, we pay costs for all vehicles that are on the road, i.e., the operational fleet size. We define the state parameter F_v, which is equal to 1 if $R_{i,v} \neq O$, i.e., when a vehicle is not located at the depot. We do not consider fuel costs or driver salaries in our cost function, since the waiting time and fleet size costs together already strike a balance between serving customers quickly, i.e., using short routes, while not unrestrictedly increasing the fleet size. Summarizing, the new cost function becomes:

$$r = h \sum_{i \in \mathcal{I}} \exp[Y_i] + f \sum_{v \in \mathcal{V}} F_v, \tag{21}$$

where f is the costs for every operational vehicle. By making Y_i exponential, high waiting times are penalized more. In our experiments, we limit the fleet size to a maximum of 100 vehicles. The decision-maker has to balance between customer waiting time and fleet costs, while anticipating future customer arrivals.

5.3 Automated Storage and Retrieval System Case

For this problem, we consider the hybrid decision space. We apply this problem variant to the routing of order-picking robots in an automated storage and retrieval system (AS/RS). Our case study is based on the system of a large Norwegian retailer (SwissLog, AutoStore, 2022a). The system as used by this retailer is implemented in many different production and distribution warehouses around the world (SwissLog, AutoStore, 2022c). The system of the Norwegian retailer contains a large grid structure on which several order-picking robots operate, see Figure 4 for an illustration of a 10×10 AS/RS system with 5 layers on which 5 robots operate and serve 4 goods-to-person workstations.

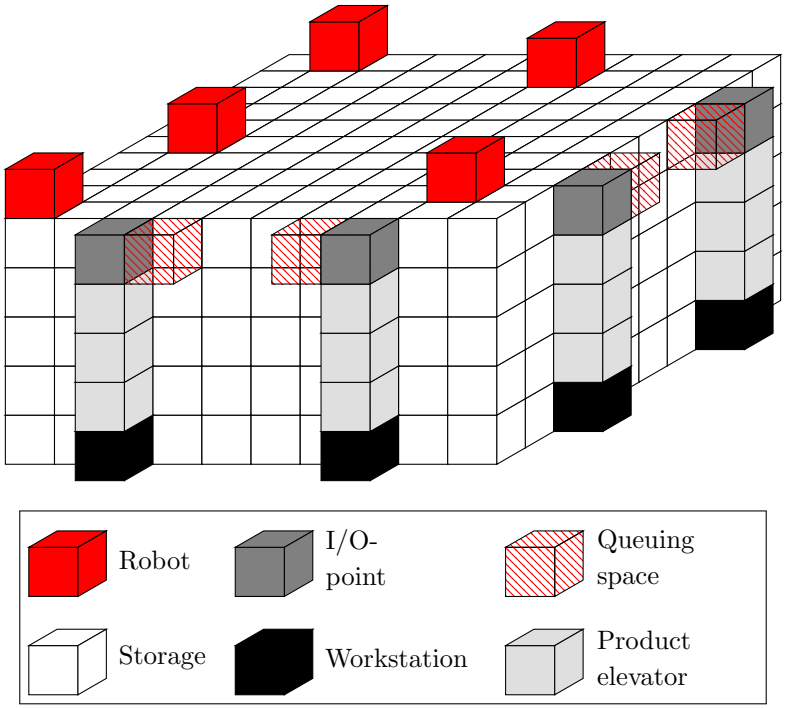


Figure 4: Illustration of an AS/RS system with a 10×10 grid with 5 layers, 5 picking robots and 4 workstations.

The robots operate on a four-directional rail that is situated on top of the shelves. These shelves

contain products that can be picked up by the robots. Robots have capacity for carrying a single product. Whenever a retrieval order is entered in the system, a robot is assigned to pick up the corresponding product at a storage location and drop it off at the right I/O-point. We consider a system where the product assortment is large and variance in the demand is high, i.e., products are stored at random locations. The I/O-point and the workstation of the human operator are connected by a product elevator. Since the elevator can only transport single products and requires some time for transporting goods, a queue of robots might form whenever the demand is high. Therefore, a queuing space outside the grid is created. In this space, a robot can either choose to wait in queue or temporarily store the picked product in order to continue executing other tasks. Nevertheless, the product still needs to be transported from the queuing area to the I/O-point when the elevator becomes available. We assume infinite capacity for storing products in the queuing area.

We only consider retrieval and do not consider storage of products. When robots are not transporting products, they can strategically reposition themselves to anticipate future orders. The goal is to maximize throughput of products on an infinite horizon. Anticipatory decision-making can be achieved in three ways: by repositioning when idle, by prioritizing orders that are requested by workstations with a smaller queue, or by dequeuing to retrieve products instead of waiting in queue. In the remainder of this section, we provide the changes to the problem formulation in Section 3.

The state is altered by adding the pickup and delivery locations P_i and D_i, respectively, a binary variable k_v indicating whether a robot carries a product, a binary variable $u_{v,w}$ indicating whether a robot is waiting in the queue at workstation $w \in \mathcal{W}$, and the length of a queue and the number of products in temporary storage at a queuing space, Q_w and B_w, respectively:

$$\mathcal{S} = \Big[(R_{i,v}, k_v, \mathcal{G}_v)_{\forall v \in \mathcal{V}}, (P_i, D_i)_{\forall i \in \mathcal{I}}, (Q_w, B_w)_{\forall w \in \mathcal{W}}, (u_{v,w})_{\forall v \in \mathcal{V}, \forall w \in \mathcal{W}}\Big].$$

The decision is to assign newly arrived tasks to a robot and, if applicable, reposition the idle robots. That is:

$$a = \Big[a^T, (a_v^R, a_v^Q)_{\forall v \in \mathcal{V}}\Big],$$

where $a^T \in \mathcal{V} \cup \{\varnothing\}$ indicates to what robot the new task is assigned, if not rejected, a_v^R is the new location of an idle robot, and a_v^Q is a binary variable indicating whether a robot will dequeue. Several constraints, omitted for conciseness, ensure that a_v^R and a_v^Q are only relevant whenever the robot is idle or queuing, respectively. We ensure that the task of driving to location P_c is executed before transporting a product to the corresponding workstation location D_c. After the assignment or rejection decision a^T, a task is added to the task list $\mathcal{G}_v$ of the respective robot. Robots execute the tasks on their task list in a first-in-first-out (FIFO) sequence. Whenever a robot dequeues, a new task is created for transporting the product from the temporary storage to the I/O-point. The path planning of robots is considered an external process and collisions of robots are not considered. After making the decision and the execution of one step in the routing plan, waiting costs h are paid. This constitutes the cost function: $r = h \sum_{i \in \mathcal{I}} P_i$.

For this case study, we base our instance settings on data from a Norwegian retailer (SwissLog, AutoStore, 2022a). We consider a grid structure of 30×30 with a single layer of shelves. These types of systems normally have between 4 and 24 layers (SwissLog, AutoStore, 2022c), but for typical

retailers using this type of system, approximately 80% of the products can be retrieved from the top layer (SwissLog, AutoStore, 2022b). On the grid structure, 15 robots operate and serve 4 workstations, located at two sides of the grid, similar to the situation depicted in Figure 4. The elevator service time is equal to 3 time periods, i.e., only after three decision epochs, an elevator is available for transporting a product again. Table 2 summarizes the problem settings. The first instance has low demand intensity: after a decision epoch new demand will appear with 50% chance. The second instance has higher demand intensity: after every decision epoch always new demand appears. We assume that these fluctuations in demand intensity can be predicted fairly easily, i.e., we train the policy for different demand intensity scenarios. However, the human operators at the four workstations might cause fluctuations in demand, e.g., caused by different working pace, non-synchronous breaks, or non-availability. Therefore, we conduct robustness experiments to observe if our policies, trained on instances with uniform demand from the 4 workstations, can deal with a different demand distribution from the 4 workstations, without retraining. Here, most demand comes from 2 workstation, while only a faction of the orders comes from the other 2 workstation. We study the robustness of different policies, since learning-based policies are prone to decrease in performance when evaluated on different instances compared to the training instances (Pinto et al., 2017; Waubert de Puiseau et al., 2022).

Table 2: Experimental settings for the robotic AS/RS case study.

Grid size $l \times l$	Number of robots v	Demand intensity	Workstation demand distribution
30×30	15	0.5	Uniform
30×30	15	1.0	Uniform
30×30	15	0.5	{0.4,0.1,0.1,0.4}
30×30	15	1.0	{0.4,0.1,0.1,0.4}

6 Computational Experiments

We conduct experiments with the policies presented in Section 4, i.e., linear VFA (LVFA), linear PFA (LPFA), neural network VFA (NNVFA), and neural network PFA (NNPFA). Normally, linear policies require engineered features ϕ and deep neural networks should find their own representation of the state space. However, to make the comparison fair, we first supply all policies with the handcrafted features. Next, we study what happens when we only supply the state to the neural network policies, i.e. ψ. All problems and algorithms were implemented in C++, using DynaPlex (Akkerman et al., 2022). The neural network models are implemented in LibTorch, a C++ library version of PyTorch (Paszke et al., 2019). All computational experiments where processed on a high performance cluster of up to 3 nodes. The hyperparameter settings are provided and more hardware details are provided in the appendix. We compare the policies using a myopic benchmark heuristic, which is explained in the appendix. We show the performance of the policies on all instances using a long simulation horizon (500 decision epochs) with 10,000 replications. We determine the number of replications such that the coefficient of variation of the downstream costs over the replications is $< 10\%$ for all experiments. Initial states are randomly

sampled. We present the results for the stylized problem in Section 6.1, the same-day pickup and delivery case in Section 6.2, and the AS/RS case in Section 6.3.

6.1 Results for the Stylized Problem

In Table 3, we report the following statistics that show the performance of the policies and indicate their anticipatory behavior. DIST VEH is the average cumulative distance between vehicles over the simulated horizon, this statistic indicates how well the vehicles spread over the grid. DIST RETURN is the average distance required to return to the depot whenever a vehicle has no remaining capacity. This statistic indicates if a policy is able to anticipate that a vehicle has to return to the depot soon and therefore already returns early. DEPOT VISITS indicates the average number of depot visits per vehicle, SIGMA ASSIGN is the standard deviation of the number of customer assignments made over the vehicles, i.e., this statistic is an indicator of the spread of the workload. DEVIATE indicates the average number of times the policy deviated from the route (only for the hybrid decision space). Finally, SAVE indicates the percentage cost savings compared to a myopic policy and SIGMA is the standard deviation of the saving, expressed as a percentage of the average saving. First, we discuss the 1-step decision variant. We observe large differences between the policies, although all policies are able to outperform the myopic benchmark. The LVFA and LPFA find a good policy that outperforms the myopic benchmark. Overall both policies perform quite similarly in terms of the average distance between vehicles, the distance to the depot upon reaching the maximum vehicle capacity, and the number of depot visits. Next, we observe that both NNVFA and NNPFA, which are supplied with the same features, outperform the linear policies. This is reflected by the spread of the vehicles over the grid, the distance upon returning, and the number of depot visits. The NNPFA outperforms the NNVFA. Finally, we do a study by removing the engineered features from the feature set and only supplying the state-literal features ψ, see Section 4.2. Interestingly, we observe that the NNVFA decreases in performance, which indicates the value of engineered features for the neural network. However, the NNPFA increases in performance. Numerous reasons can account for this, but most probably the engineered features have good predictive value for the NNVFA but hinder the NNPFA in finding more "creative" policies, since it is only supplied with partial state information. Another explanation could be that the engineered features cause some noise in observations, for which the NNPFA might be more vulnerable.

Next, we consider the assignment decision space. We observe that all policies have more trouble to outperform the myopic policy, perhaps caused by the simpler decision space, from which the myopic policy also benefits. We observe that the distribution of vehicles over the grid is better compared to the 1-step decision. Additionally, the distance upon returning increased, but the number of depot visits increased, which indicates that the vehicles are able to serve more customers. The largest difference between the policies can be observed in the standard deviation over the assignments per vehicle. Overall, we conclude that, similar to the 1-step decision, the linear policies are outperformed by the neural network policies, and the NNPFA has en edge over the NNVFA. Again, the study with the feature sets ϕ and ψ shows similar results. As an additional experiment with the assignment decision, we study how the policies perform on a larger instance, namely a 20×20 grid with 10 vehicles. Interestingly, we now see a

Table 3: Results for the stylized problem.

	DIST VEH	DIST RETURN	DEPOT VISITS	SIGMA ASSIGN	DEVIATE	SAVE (%)	SIGMA (%)
1-STEP DECISION							
LVFA	41.0	6.2	8.7	-	-	10.8%	0.8%
LPFA	45.0	5.9	8.6	-	-	6.2%	0.7%
NNVFA ϕ	50.6	5.3	15.5	-	-	14.6%	1.0%
NNPFA ϕ	42.3	3.8	20.3	-	-	21.2%	0.8%
NNVFA	49.9	2.9	19.4	-	-	11.4%	0.8%
NNPFA	48.5	3.2	28.5	-	-	25.6%	1.1%
ASSIGNMENT DECISION							
LVFA	144.1	9.2	9.2	101.0	-	2.6%	0.4%
LPFA	135.0	9.1	9.2	88.1	-	3.4%	0.5%
NNVFA ϕ	136.3	9.5	12.4	61.4	-	11.7%	0.9%
NNPFA ϕ	137.1	10.4	8.2	149.2	-	4.2%	1.6%
NNVFA	133.0	10.1	11.8	76.9	-	4.1%	1.2%
NNPFA	134.8	8.9	4.9	59.2	-	13.9%	1.3%
ASSIGNMENT DECISION 20×20							
LVFA	224.1	15.2	6.0	78.0	-	28.6%	0.8%
LPFA	219.0	14.5	5.9	89.1	-	25.8.4%	1.1%
NNVFA	245.0	16.1	4.5	175.9	-	14.3.1%	1.2%
NNPFA	234.8	15.8	5.1	155.5	-	20.9%	0.9%
HYBRID DECISION							
LVFA	157.0	9.8	8.9	101.0	10	2.7%	0.5%
LPFA	122.4	10.9	7.6	123.1	2	3.5%	0.3%
NNVFA ϕ	131.3	9.1	9.1	59.4	38.9	10.8%	1.7%
NNPFA ϕ	130.2	10.2	9.1	145.2	22.2	4.9%	1.1%
NNVFA	127.1	10.0	8.7	75.2	40.6	3.2%	1.3%
NNPFA	128.2	9.2	9.4	61.2	39.2	17.2%	2.1%

completely different result: LVFA is the best policy, closely followed by LPFA. The NNPFA and NNVFA policy seem to struggle with the larger state and decision space. We observe that the costs savings of the linear policies are mainly stemming from the better distribution of customers over vehicles, as the SIGMA ASSIGN statistic indicates.

Finally, the hybrid decision shows that the linear policies hardly deviate from the planned route. The neural network policies however, make more deviations from the planned route. The possibility of deviating from the planned route does not always seem to yield lower costs; only for the NNPFA we observe a significant cost decrease. In Figure 5, we show the decision made by four policies on a hand-picked state. This state is a representative example for the states observed during training. This state includes 5 vehicles with different capacities, where 1 vehicle is idle. The policy needs to decide on the assignment of customer C_8 to a vehicle route, and on the possible deviation from a route, if applicable. First, we see that all policies show anticipatory behavior since they are able to identify that the top left vehicle needs to return to the depot soon, and therefore it deviates from the route. The idle vehicle on the right side is only moved by the neural network policies. We see that the LVFA is the only policy that consider to directly serve customer C_8 by deviating with the closest-by vehicle, this is the same decision the myopic policy makes. All other policies assign the new customer to the end of a route. Furthermore, the NNPFA makes an interesting, "creative" decision: it decides to deviate and move to the left and serve a customer that is close but already assigned to a different vehicle.

There are large differences between the algorithms for obtaining the required sample size. The algorithm for the LVFA policy finishes training within 8 minutes (1 node), the LPFA algorithm requires 4 minutes of training, the algorithm for the NNVFA policy requires 1 hour of training time on 3 nodes, and the algorithm for the NNPFA policy requires almost 9 hours of training time on 3 nodes.

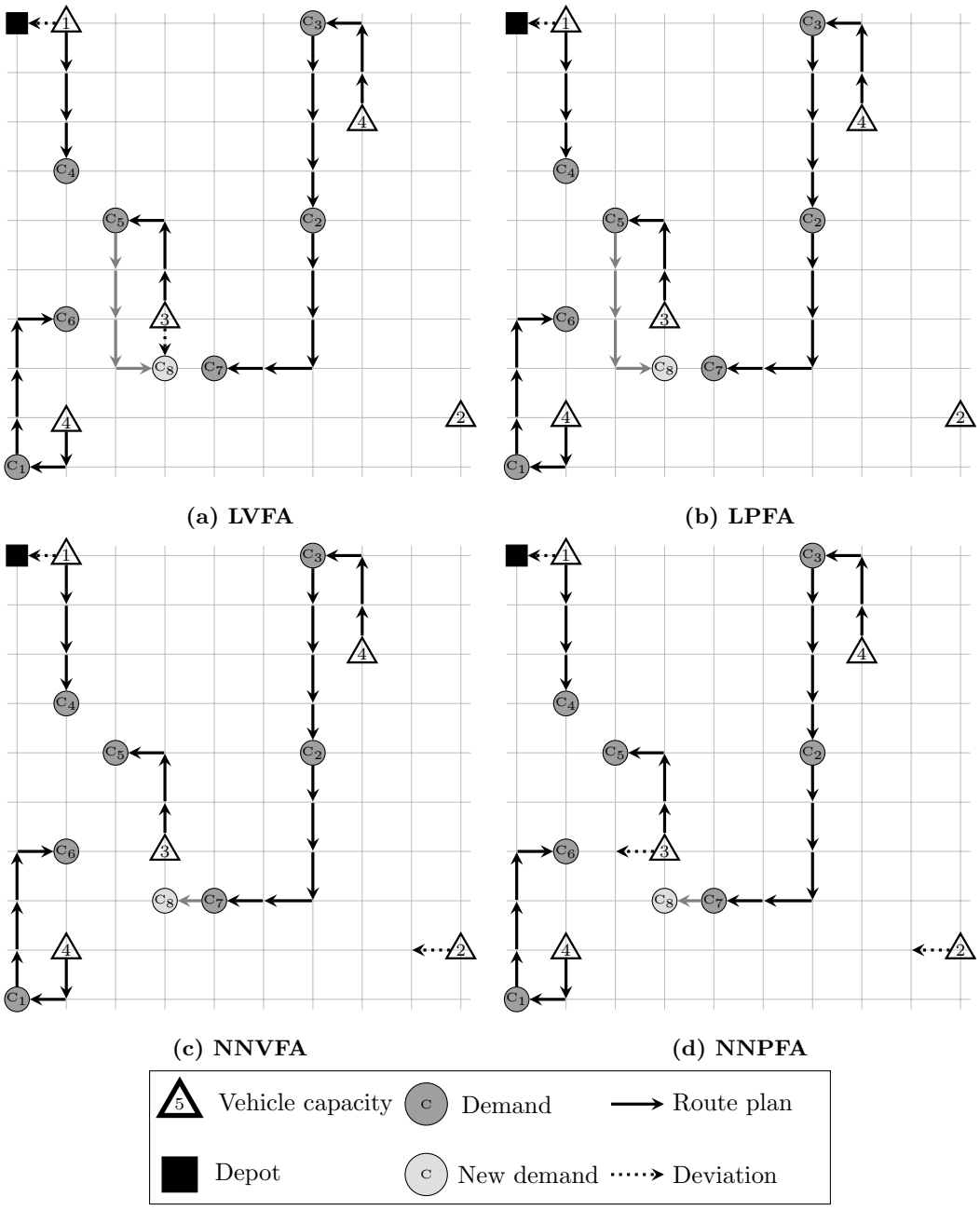


Figure 5: Decisions made under different policies for the hybrid decision space.

6.2 Results for the Same-day Parcel Pickup and Delivery Case

Table 4 shows the results for the parcel pickup and delivery case in the city of Amsterdam. We report the average distance travelled per vehicle, the average customer waiting time in minutes, the average number of utilized vehicles, and the total costs. For this experiment, we supply the linear policies with the engineered features ϕ, and the neural network policies are supplied with the state literal ψ. First, we observe that the myopic policy is not able to strike a balance between minimizing waiting time and the fleet size, i.e., the fleet size is kept to a minimum, which causes longer waiting times, and hence, higher costs compared to all other policies. The performance of all policies is relatively close together, with the neural network policies slightly outperforming the linear policies. The NNPFA shows the best results in

terms of costs.

Table 4: Performance of policies for the same-day parcel pickup and delivery case.

Policy	Km per vehicle	Avg. customer waiting time (min.)	Nr. of utilized vehicles	Total costs
Myopic	145.2	49	16	3,763
LVFA	100.2	32	24	2,953
LPFA	98.2	32	25	3,056
NNVFA	89.2	29	28	2,921
NNPFA	91.1	29	27	2,886

6.3 Results for the Automated Storage and Retrieval System Case

Figure 6 shows the percentage saving of the four policies compared to the benchmark myopic policy. On the instance with lower demand intensity, we see that all policies can beat the benchmark. LVFA especially finds a good policy with a 27.3% saving, closely followed by the LPFA and NNPFA that save 25.3% and 25.5%, respectively. These results are remarkable in comparison to the previous results. It confirms that the relative performance among policies is highly problem-dependent. We observe that the repositioning and dequeuing decisions allow for large savings compared to the benchmark, which does not reposition or dequeue. Results for the High Uniform Demand instance confirm this: when demand is high, all robots always have tasks, and all I/O-points have queues, repositioning and dequeuing decisions have little to no effect, and no policy is significantly better compared to the benchmark. The robustness experiments show that all policies are relatively robust for changes in the workstation demand distribution; the NNVFA method shows the smallest decrease in performance.

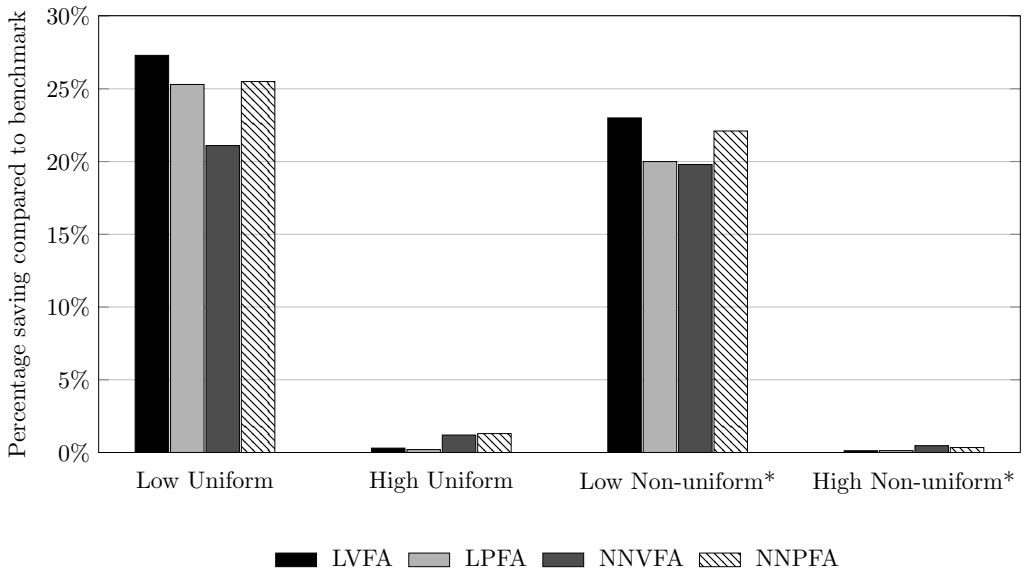


Figure 6: Savings of prescriptive policies for the AS/RS case, compared to the benchmark. *robustness experiments.

7 Discussion and Conclusion

In this paper, we study the use of different classes of reinforcement learning, with and without the use of neural networks, applied to dynamic vehicle routing problems (DVRPs) with stochastic customer requests. We provide a comparison of a linear value function approximation (VFA), linear policy function approximation (LPFA), neural network VFA (NNVFA), and neural network PFA (NNPFA). We show that the four studied policies are rather similar and can be applied to the same DVRPs, hence, it is valuable to conduct a comparison. We study the policies' relative performance using different stylized problem variants of the dynamic vehicle routing problem with stochastic customer requests. We validate our results on real-world cases that are extensions of the stylized problem. With our experiments, we hope to shed light on the advantages and disadvantages of neural network-based policies for DVRPs compared to the prevailing linear policies. Our comparison yields four findings: (i) the relative performance of neural network-based policies compared to linear policies is highly dependent on problem characteristics, (ii) the ability of neural networks to improve upon linear policies is drawn from the ability to capture the complex relationship between the state variables and downstream costs of the state, including interaction effects. However, (iii) this comes at the expense of considerably longer computational times. Additionally, we show that (iv) NNPFAs most often outperform NNVFAs, but NNVFAs provide a significantly faster alternative.

We show that all anticipatory policies can perform significantly better on all problem instances compared to the myopic benchmark. We conclude that subtle modifications of the problem size might require a different learning architecture, and it might not be clear up front which design provides the best performance. Overall, we observe that slight modifications in problem elements suddenly favor different policies. In many cases, the NNPFA outperforms the LVFA, LPFA, and NNVFA methods, but for a large instance, the linear methods improve compared to the neural networks and even outperform them.

Linear policies can be a powerful method, with low computational demands, high sample efficiency, and high interpretability. When linear structures in an MDP can be easily exploited, i.e., when features with a linear relationship to costs are self-evident, linear regression is a good starting point for stochastic optimization policies. We observed that engineered features can have a positive influence on the performance of neural network policies. However, the use of non-engineered features, i.e., just supplying the state, has major advantages: no expert problem knowledge is needed for creating features and the neural network can infer their own representation of the value of states. This can result in more "creative" anticipatory policies, like shown in Figure 5.

Based on our insights, we provide several actionable recommendations to researchers and practitioners in DVRP. First, we advise to always start with simple, interpretable decision-making policies, i.e., a linear policy that employs linear basis functions. When the engineering of features is difficult, we advise to leverage the power of neural networks. Rather than supplying engineered features to the neural networks, they should be provided with only the state and find their own feature representation of the state space. However, we advise to tests if neural network performance can be improved by supplying engineered features. Training neural network-based policies require more computational effort since

neural networks need more data to be trained and the training process itself is more demanding. When the simulation/sampling is relatively cheap, neural networks are an excellent method that can learn complex relationships and require little to no expert problem knowledge.

Further research can be done on the different dimensions of design choices for RL. Many design choices need to be made, e.g., the MDP formulation, policy selection, algorithm selection, and tuning. It is often poorly understood what the effect of these choices is on the performance of RL for transportation problems. Additionally, different prediction models can be applied to VFA and PFA policies, e.g., random forests, logistic regression, convolutional neural networks, or graph neural networks. Finally, the application of ensemble methods that combine learning-based policies into a single policy might be an interesting research direction for DVRPs, since it is often unknown upfront what approach works best.

CRediT authorship contribution statement

Fabian Akkerman Conceptualization, Investigation, Software, Writing – original draft.
Martijn Mes Conceptualization, Supervision, Writing – original draft.
Willem van Jaarsveld Conceptualization, Software, Supervision, Writing – original draft.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data and software will be made available on request.

References

Fabian Akkerman and Martijn Mes. Distance approximation to support customer selection in vehicle routing problems. *Annals of Operations Research*, Apr 2022.

Fabian Akkerman, Luca Begnardi, Riccardo Lo Bianco, Tarkan Temizoz, Remco Dijkman, Maria Iacob, Martijn Mes, Yingqian Zhang, and Willem van Jaarsveld. DynaPlex, 2022. URL `http://www.dynaplex.nl`. Accessed October 3, 2022.

Itamar Arel, C. Liu, T. Urbanik, and Airton Kohls. Reinforcement learning-based multi-agent system for network traffic signal control. *Intelligent Transport Systems, IET*, 4:128 – 135, 07 2010.

Florian Arnold and Kenneth Sörensen. What makes a vrp solution good? the generation of problem-specific knowledge for heuristics. *Computers & Operations Research*, 106:280–288, 2019. ISSN 0305-0548.

Amin Asadi and Sarah Nurre Pinkley. A monotone approximate dynamic programming approach for the stochastic scheduling, allocation, and inventory replenishment problem: Applications to drone and electric vehicle battery swap stations. *Transportation Science*, 56(4):1085–1110, 2022.

Bharathan Balaji, Jordan Bell-Masterson, Enes Bilgin, Andreas Damianou, Pablo Moreno Garcia, Arpit Jain, Runfei Luo, Alvaro Maggiar, Balakrishnan Narayanaswamy, and Chun Ye. Orl: Reinforcement learning benchmarks for online stochastic optimization problems, 2019.

Breno A. Beirigo, Frederik Schulte, and Rudy R. Negenborn. A learning-based optimization approach for autonomous ridesharing platforms with service-level contracts and on-demand hiring of idle vehicles. *Transportation Science*, 56(3):677–703, 2022.

Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-dynamic programming.* Optimization and neural computation series. Athena Scientific, Belmont, MA, USA, 1996.

Dimitris Bertsimas and Ramazan Demir. An approximate dynamic programming approach to multidimensional knapsack problems. *Management Science*, 48(4):550–565, 2002.

Robert N. Boute, Joren Gijsbrechts, Willem van Jaarsveld, and Nathalie Vanvuchelen. Deep reinforcement learning for inventory control: A roadmap. *European Journal of Operational Research*, 298(2): 401–412, 2022.

Belgacem Bouzaiene-Ayari, Clark Cheng, Sourav Das, Ricardo Fiorillo, and Warren B. Powell. From single commodity to multiattribute models for locomotive optimization: A comparison of optimal integer programming and approximate dynamic programming. *Transportation Science*, 50(2):366–389, 2016.

Yuxuan Dong, René De Koster, Debjit Roy, and Yugang Yu. Dynamic vehicle allocation policies for shared autonomous electric fleets. *Transportation Science*, 56(5):1238–1258, 2022.

Omar El Housni and Vineet Goyal. On the optimality of affine policies for budgeted uncertainty sets. *Mathematics of Operations Research*, 46(2):674–711, 2021.

Matthieu Geist and Olivier Pietquin. Algorithmic survey of parametric value function approximation. *IEEE Transactions on Neural Networks and Learning Systems*, 24(6):845–867, 2013.

Joren Gijsbrechts, Robert N. Boute, Jan A. Van Mieghem, and Dennis J. Zhang. Can deep reinforcement learning improve inventory management? performance on lost sales, dual-sourcing, and multi-echelon problems. *Manufacturing & Service Operations Management*, 24(3):1349–1368, 2022.

Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* O'Reilly Media, Inc., 1st edition, 2017.

Fang He, Jie Yang, and Meng Li. Vehicle scheduling under stochastic trip times: An approximate dynamic programming approach. *Transportation Research Part C: Emerging Technologies*, 96:144–159, 2018.

Florentin D. Hildebrandt, Barrett W. Thomas, and Marlin W. Ulmer. Opportunities for reinforcement learning in stochastic dynamic vehicle routing. *Computers & Operations Research*, 150:106071, 2023. ISSN 0305-0548.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems!, 2019.

Nicholas D. Kullman, Martin Cousineau, Justin C. Goodson, and Jorge E. Mendoza. Dynamic ride-hailing with electric vehicles. *Transportation Science*, 56(3):775–794, 2022.

Enming Liang, Kexin Wen, William H. K. Lam, Agachai Sumalee, and Renxin Zhong. An integrated reinforcement learning and centralized programming approach for online taxi dispatching. *IEEE Trans Neural Netw Learn Syst*, 33(9), 2021.

Yi Ma, Xiaotian Hao, Jianye HAO, Jiawen Lu, Xing Liu, Xialiang Tong, Mingxuan Yuan, Zhigang Li, Jie Tang, and Zhaopeng Meng. A hierarchical reinforcement learning based optimization framework for large-scale dynamic pickup and delivery problems. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

Chao Mao, Yulin Liu, and Zuo-Jun (Max) Shen. Dispatch of autonomous vehicles for taxi services: A deep reinforcement learning approach. *Transportation Research Part C: Emerging Technologies*, 115: 102626, 2020.

MohammadReza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takac. Reinforcement learning for solving the vehicle routing problem. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31, 2018.

Daniel Nicola, Rudolf Vetschera, and Alina Dragomir. Total distance approximations for routing solutions. *Computers & Operations Research*, 102:67–74, 2019. ISSN 0305-0548.

Brenner Humberto Ojeda Rios, Eduardo C. Xavier, Flávio K. Miyazawa, Pedro Amorim, Eduardo Curcio, and Maria João Santos. Recent dynamic vehicle routing problems: A survey. *Computers & Industrial Engineering*, 160:107604, 2021.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 8–14 Dec 2019.

Dai T. Pham and Gudrun P. Kiesmüller. Multiperiod integrated spare parts and tour planning for on-site maintenance activities with stochastic repair requests. *Computers & Operations Research*, 148:105967, 2022.

Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 2817–2826. JMLR, 6–11 Aug 2017.

Warren B. Powell. A unified framework for stochastic optimization. *European Journal of Operational Research*, 275(3):795–821, 2019.

Warren B. Powell and Stephan Meisel. Tutorial on stochastic optimization in energy—part ii: An energy storage illustration. *IEEE Transactions on Power Systems*, 31(2):1468–1475, 2016.

Warrren B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality: Second Edition.* Wiley-Blackwell, Hoboken, NJ, USA, sep 2011.

Warrren B. Powell. *Reinforcement Learning and Stochastic Optimization: A Unified Framework for Sequential Decisions.* Wiley-Blackwell, Hoboken, NJ, USA, march 2022.

Warrren B. Powell and Ilya O. Ryzhov. *Optimal Learning.* Wiley-Blackwell, Hoboken, NJ, USA, 2012.

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

Zhiwei (Tony) Qin, Xiaocheng Tang, Yan Jiao, Fan Zhang, Zhe Xu, Hongtu Zhu, and Jieping Ye. Ride-hailing order dispatching at didi via reinforcement learning. *INFORMS Journal on Applied Analytics*, 50(5):272–286, 2020.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

Ninja Soeffker, Marlin W. Ulmer, and Dirk C. Mattfeld. Stochastic dynamic vehicle routing in the light of prescriptive analytics: A review. *European Journal of Operational Research*, 298(3):801–820, 2022.

SURFsara. Snellius hardware and file systems, 2022. URL `https://servicedesk.surfsara.nl/wiki/display/WIKI/Snellius+hardware+and+file+systems`. Accessed May 4, 2022.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, Cambridge, MA, USA, second edition, 2018.

SwissLog, AutoStore. Warehouse automation adds to elektroimportøren's success, 2022a. URL https://www.autostoresystem.com/cases/warehouse-automation-adds-to-elektroimportorens-success. Accessed March 22, 2022.

SwissLog, AutoStore. Autostore for retail, 2022b. URL https://www.autostoresystem.com/industries/retail. Accessed March 22, 2022.

SwissLog, AutoStore. Autostore cases, 2022c. URL https://www.autostoresystem.com/cases. Accessed March 22, 2022.

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

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

Marlin. W. Ulmer, Dirk. C. Mattfeld, and Felix Köster. Budgeting time for dynamic vehicle routing with stochastic customer requests. *Transportation Science*, 52(1):20–37, 2018.

Marlin W. Ulmer, Justin C. Goodson, Dirk C. Mattfeld, and Marco Hennig. Offline–online approximate dynamic programming for dynamic vehicle routing with stochastic requests. *Transportation Science*, 53(1):185–202, 2019.

Wouter van Heeswijk and Han La Poutré. Approximate dynamic programming with neural networks in linear discrete action spaces. *CoRR*, 2019.

Wouter van Heeswijk, Martijn Mes, and Marco Schutten. The delivery dispatching problem with time windows for urban consolidation centers. *Transportation science*, 53(1):203–221, January 2019.

Willem van Jaarsveld. Model-based controlled learning of mdp policies with an application to lost-sales inventory control. *CoRR*, 11 2020.

Constantin Waubert de Puiseau, Richard Meyes, and Tobias Meisen. On reliability of reinforcement learning based production scheduling systems: a comparative survey. *Journal of Intelligent Manufacturing*, 33(4):911–927, Apr 2022.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, May 1992.

Pengfei Yi, Santhosh K. George, Jomon Aliyas Paul, and Li Lin. Hospital capacity planning for disaster emergency management. *Socio-Economic Planning Sciences*, 44(3):151–160, 2010.

Jian Zhang and Tom Van Woensel. Dynamic vehicle routing with random requests: A literature review. *International Journal of Production Economics*, 256:108751, 2023.

Xiaonan Zhang, Jianxiong Zhang, and Xiaoqing Fan. Offline approximate value iteration for dynamic solutions to the multivehicle routing problem with stochastic demand. *Computers & Operations Research*, 146:105884, 2022.

Yue Zhou, Kaan Ozbay, Pushkin Kachroo, and Fan Zuo. Ramp metering for a distant downstream bottleneck using reinforcement learning with value-function approximation. *Journal of Advanced Transportation*, 2020, 10 2020.

A Learning Algorithms

Every learning-based policy requires a training algorithm. The chosen policy restricts the choice in algorithms. Although the used algorithms are slightly different, many similarities exist: all algorithms encompass a sample collection and parameter update phase. We describe the sample collection and parameter update phase in Section A.1. Next, we describe each algorithm in more detail, including pseudocode: in Section A.2 we provide the code for the LVFA, in Section A.3 for the NNVFA, and in Section A.4 for both the LPFA and NNPFA, which share the same training algorithm.

A.1 Sample Collection

The sample collection procedure starts with a policy π and initial state s. The algorithms collect information about the performance of the current policy π during several iterations i. All algorithms rely on Monte Carlo simulation for collecting the samples, but differ in the frequency of policy updates. For obtaining the LVFA policy, we follow the generic approximate value iteration (AVI) algorithm, as described in Powell (2011). Figure 7 depicts two iterations of the AVI algorithm. We start in a random state s_i and execute a random decision $\widetilde{a}_i$. Next, the feature values $\boldsymbol{\phi}(s_i^{\widetilde{a}})$ of the post-decision state are stored. With the sampled random event $\omega_i \in \Omega$ the transition is made to the next state. At the start of the second iteration, a single decision is simulated following the current decision policy $\pi_{\theta,i}^{\text{LVFA}}$, and the cumulative costs observed after this decision, including a bootstrap estimate, are stored as $\widehat{v}_i$. More explanation concerning bootstrap estimates can be found in, for instance, Sutton and Barto (2018) and Powell (2022).

Using neural networks inside the AVI algorithm to obtain the NNVFA policy is not straightforward. This has three main reasons: (i) to properly learn, neural networks require large data sets, (ii) training neural networks is more computationally expensive compared to updating a lookup table or regression model, and (iii) neural networks can be sensitive to noise in the data. We present a batch-AVI algorithm to overcome these challenges. We use a sampling and updating approach that is suitable for neural networks and keeps the generalizing strength and sample efficiency of VFA methods. We alter the AVI algorithm by (i) postponing VFA updates to collect a large data set of size I, (ii) simulating a longer horizon L to obtain a more reliable downstream cost estimate, and (iii) conducting replications to further improve reliability of the downstream costs and reduce noise. Figure 8 shows a single sample collection step, without replications. We start with some random state s_i and execute a random decision $\widetilde{a}_i$ to obtain a post-decision state and store its respective features $\boldsymbol{\psi}(s_i^{\widetilde{a}})$. Next, we conduct a simulation inside

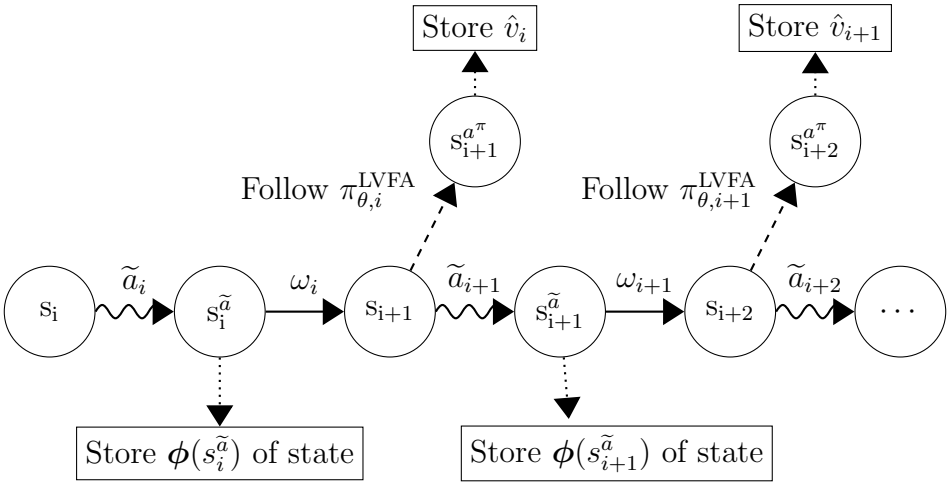


Figure 7: Sample collection for approximate value iteration

a simulation of length L and store the accumulated downstream costs $\widehat{v}_i$. Next, to obtain a new state for our next sample, we take a random decision starting from state s_{i+1}.

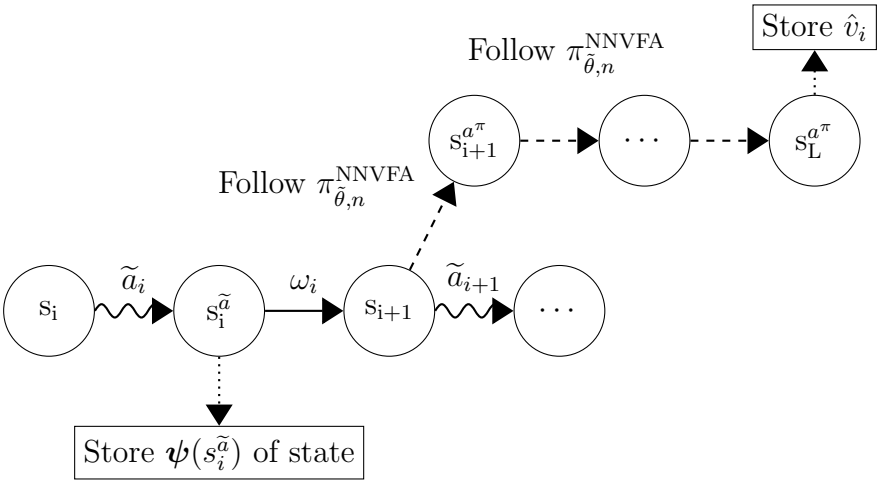


Figure 8: Sample collection for batch approximate value iteration.

In line with the batch-AVI algorithm introduced above, for learning the LPFA and NNPFA policies we adopt a batch approximate policy iteration (batch-API) algorithm. The adopted algorithm is a natural extension of the batch-AVI algorithm that has conceptual similarities to AlphaZero (Silver et al., 2018), see also van Jaarsveld (2020). For batch-API, we take a similar approach as batch-AVI, except that we estimate the downstream costs for all decisions from a state s_i, and not just for a single random decision. This allows for the identification of a decision that corresponds to the lowest estimated downstream cost (under the current policy). Again, we try to reduce noise by simulating longer trajectories and conducting replications. In Figure 9, the policy evaluation step of a single iteration i and single replication is depicted. First, a random state s_i is drawn and the feature values of this state $\boldsymbol{\psi}(s_i)$ are stored. All feasible decisions are executed on parallel trajectories; in our example, we have three feasible decisions $\{a^1, a^2, a^3\}$. For each feasible decision, trajectories are followed using the current policy $\pi_{\bar{\theta},n}^{\text{PFA}}$ and fixed sample path $\omega \in \Omega$. At the end of such a sampled trajectory, we end up in the state s_L and can evaluate

the downstream costs $\mathcal{Z}_a$ of the trajectories corresponding with each initial decision a_i. We store the decision $\widehat{a}_i$ that returns minimum $\mathcal{Z}$.

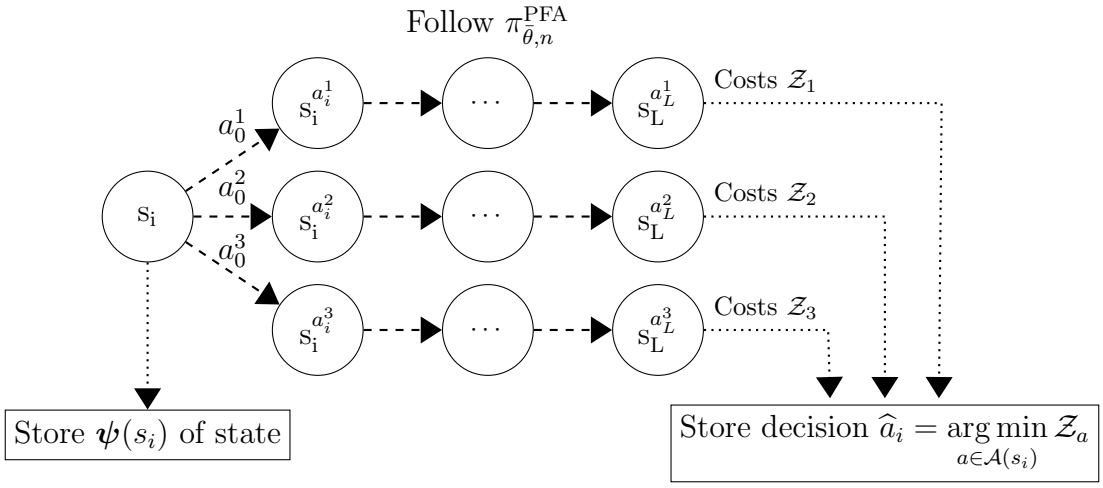


Figure 9: Sample collection for batch approximate policy iteration.

Note that for all algorithms, we obtain new samples of initial states by executing random decisions $\widetilde{a}$ on the previously observed states. However, for some problems, making random decisions might result in sampling states that are undesirable. Hence, different sampling strategies might be needed as opposed to making random decisions, e.g., ϵ-greedy or softmax decision selection. These strategies balance the exploration versus exploitation trade-off; more examples and explanations can be found in the optimal learning literature, see, for instance, Powell and Ryzhov (2012).

A.2 Approximate Value Iteration

Algorithm 1 provides a high-level overview of approximate value iteration, as used for training the LVFA. This version of the algorithm uses a direct update by a forward pass in combination with pure exploration (Powell, 2011). We denote a value function approximation by $\overline{V}$, a sequence of random events (sample path) by $\omega \in \Omega$, and the downstream costs by $\hat{v}$. Note that the approximated downstream costs are included in $\hat{v}$, i.e., we use a bootstrap estimate (Sutton and Barto, 2018). Approximate value iteration requires regular convergence checks since the extracted policy $\widehat{\pi}$, i.e., greedily selecting the decision with lowest expected discounted costs, might has converged whereas the value function $\overline{V}$ has not.

Algorithm 1: Approximate Value Iteration

Step 0.	**Initialization**	
	Step 0a.	Initialize the approximation for $\overline{V}_0$
	Step 0b.	Set the iteration counter to $n = 0$, and number of iterations N
Step 1.	Choose a sample path $\omega \in \Omega$ and initial state s_0	
Step 2.	**Policy Extraction and Value Function Updating**	
	Step 2a.	If $n > 0$, find feasible decision a_n that solves $\hat{v}_{n-1} = \min_{a \in \mathcal{A}(s_n)} C(s_n, a) + \gamma \overline{V}_{n-1}(s_n^a)$
	Step 2b.	Transition to post-decision state $s_n^{\widetilde{a}}$ with random decision $\widetilde{a}_n$
	Step 2c.	Store feature values $\boldsymbol{\phi}_n$ of state $s_n^{\widetilde{a}}$
	Step 2d.	Transition to state s_{n+1} with sampled event ω_n
	Step 2e.	If $n > 0$, **update approximation** $\overline{V}_n \leftarrow \overline{V}_{n-1}$ with $\boldsymbol{\phi}_{n-1}$ and $\hat{v}_{n-1}$
Step 3.	Increment n, if $n \leq N$ go to Step 2, else return $\overline{V}_N$	

A.3 Batch Approximate Value Iteration

Algorithm 2 provides a high-level overview of the batch approximate value iteration approach. It is in essence the same as the standard approximate value iteration algorithm (see Algorithm 1), with the following alterations: (i) for the simulation after a decision we use a longer simulation trajectory L, instead of the single simulation step as common for textbook approximate value iteration, (ii) we postpone the value function update until we collect I samples, and use the batch of features Φ_n and corresponding end-of-horizon costs $\mathcal{H}_n$ for the update, and (iii) we conduct replications of Step 2a to Step 2b to obtain a more reliable estimate of the end-of-horizon costs $H_i \in \mathcal{H}_n$. Note that we only include a bootstrap estimate at the end of the simulation horizon L (Step 2biii in Algorithm 2). When using a longer sampling horizon L, costs are discounted with a polynomial discount factor γ^l, with l being the current step in the horizon of length L and $\gamma < 1$. For brevity, we exclude the use of replications in Algorithm 2.

Algorithm 2: Batch Approximate Value Iteration

Step 0. **Initialization**

- Step 0a. Initialize the approximation for $\overline{V}_0$
- Step 0b. Set the iteration counter to $n = 0$, and number of iterations N
- Step 0c. Set the simulation trajectory length L and number of samples I

Step 1. Set the sample size counter to $i = 0$

Step 2. **Sampling**

- Step 2a. Choose sample path $\omega_i \in \Omega$ and sample state s_0
- Step 2b. **Do for** $l = 1$ **to trajectory length** L
  - Step 2bi. Find feasible decision a_l that solves
    $$a = \arg\min_{a \in \mathcal{A}(s_l)} C(s_l, a) + \gamma^l \, \overline{V}_n(s_l^a)$$
  - Step 2bii. If $l < L$:
    $$H_i \leftarrow H_i + C(s_l, a)$$
  - Step 2biii. If $l = L$:
    $$H_i \leftarrow H_i + C(s_l, a) + \gamma^l \, \overline{V}_n(s_l^a)$$
- Step 2c. Transition to post-decision state $s_n^{\widetilde{a}}$ with random decision $\widetilde{a}_i$
- Step 2d. Store feature values $\boldsymbol{\phi}_i$ of state $s_i^{\widetilde{a}}$
- Step 2e. Transition to new state s_{i+1} with sampled event ω_i
- Step 2f. Increment i, if $i \leq I$ go to Step 2a, else continue to Step 3

Step 3. **Value Function Updating**

- Step 3a. Update approximation $\overline{V}_{n+1} \leftarrow \overline{V}_n$ with the sets Φ_n and $\mathcal{H}_n$

Step 4. Increment n, if $n \leq N$ go to Step 1, else return $\overline{V}_N$

A.4 Batch Approximate Policy Iteration

Algorithm 3 provides a high-level overview of the approximate policy iteration algorithm, which shares characteristics with the batch-AVI algorithm from Section A.3. We denote the total discounted costs at the end of a trajectory by $\mathcal{Z}$. Note that for conciseness, we exclude the use of replications in Algorithm 3. In our experiments, we conduct replications of Step 2a to Step 2c to obtain a more reliable estimate of $\mathcal{Z}_a$ and the best scoring decision $\widehat{a}_i \in \mathcal{A}(s_i)$.

Algorithm 3: Batch Approximate Policy Iteration

Step 0. **Initialization**

- Step 0a. Initialize the base policy π_0
- Step 0b. Set the iteration counter to $n = 0$, and number of iterations N

Step 1. Set the sample size counter to $i = 0$, and required number of samples I

Step 2. **Approximate Policy Evaluation**

- Step 2a. Choose sample path $\omega_i \in \Omega$ and sample state s_0
- Step 2b. **Do for all feasible decisions** $a_i \in \mathcal{A}(s_0)$
  - Step 2bi. Transition to next state s_1 with decision a_i
  - Step 2bii. Find downstream costs $\mathcal{Z}_a$ with $\widehat{\pi}_n$ and sample path ω_i

$$\mathcal{Z}_a(s_i, a, \widehat{\pi}_n, \omega_i) = \sum_{l=1}^{L} C(s_i, \widehat{\pi}_n, \omega_{i,l})$$

- Step 2c. Store decision $\widehat{a}_i \in \underset{a_i \in \mathcal{A}(s_i)}{\arg\min}\, \mathcal{Z}_a$ and feature values $\boldsymbol{\phi}_{n,i}$ of state s_i
- Step 2d. Increment i, if $i \leq I$ go to Step 2a, else continue to Step 3

Step 3. **Policy Improvement**

- Step 3a. Transition to new policy $\widehat{\pi}_{n+1}$ using I stored samples

Step 4. Increment n, if $n \leq N$ go to Step 1, else return $\widehat{\pi}_N$

B Parameter Settings

For each policy, we start from scratch, i.e., we obtain the first samples using random decisions. Samples of states are randomly generated, using pure exploration. Especially for LVFA this causes some instability in the parameter updates. We manage instability during training by often resetting (after 25 decision epochs) to a relatively empty initial state. This way, the sampled states are slightly more uniform and the algorithm is more stable. For all algorithms, the initial state is randomly generated by drawing random initial vehicle locations and drawing a random number of initial customers. The initial number of customers is drawn from a discrete uniform distribution with bounds $U[1, l]$, where l is the dimension parameter for the $l \times l$ grid.

For all cases and algorithms, we use the same hyperparameter settings, which are tuned on a single representative instance. The neural network architecture has three fully connected hidden layers with each 128 hidden nodes. The minibatch size for training is 64. For the neural network-based policies, we ensure that samples are reliable by conducting sufficiently many replications per sample (cf. van Jaarsveld, 2020). The simulation trajectory parameter L is tunable and especially important for the VFA policies. After tuning, we find that for LVFA and NNVFA, $L = 1$ and $L = 2$ strike the best balance between performance and computational time, respectively. Longer horizons are simulated for NNPFA; the length of the horizon for this algorithm is drawn from a geometric distribution with an expected length of 100, $L \sim \text{Geo}(0.01)$. Therefore, sampling is significantly more computationally demanding compared to the VFA methods. For all policies, we use a discount factor γ of 0.99 during training.

C Benchmark Myopic Policies

Since even small instances are intractable to solve exact, we benchmark our policies with a heuristic. We compare with a heuristic that shows acceptable performance on the respective problems. For the 1-step decision problem as defined in Section 3, we use the following rule to decide on each vehicles' next location:

$$a_{=} \underset{\forall a \in a_v, \forall i \in \mathcal{I}}{\arg\min} d(a, D_i) \implies D_i = 1, \qquad \forall v \in \mathcal{V}. \tag{22}$$

Thus, the vehicle moves in the direction of the closest customer D_i. In case of ties, the decision is selected randomly among the equally valued options.

For the assignment problem variant as presented in Section 3, we need to assign the new customer demand, denoted by w, to a vehicle $v \in \mathcal{V}$. The benchmark assigns a customer to a vehicle that has an already assigned customer in the set $\mathcal{G}_v$ closest to the new customer:

$$a = \underset{\forall v \in \mathcal{V}}{\arg\min} \left(\min_{\forall i \in \mathcal{G}_v} \{d(i, w)\} \right). \tag{23}$$

To avoid unequal distribution over vehicles, we also consider the current vehicle location $R_{i,v}$ for calculating the distances, and whenever a vehicle is idle, i.e., has zero customers in its customer list ($|\mathcal{G}_v| = 0$), the new customer is directly assigned to the respective vehicle. Whenever there are multiple idle vehicles, the customer is assigned to the closest vehicle. The decision rule as used for the AS/RS case as given in Section 5.3 is the same as used for the dynamic assignment problem, see Equation 23.

D Computational Hardware

Computations are conducted on 1 or 3 thin CPU nodes of the Snellius high performance cluster (SURFsara, 2022). Algorithms that can run in parallel, i.e., batch-AVI (NNVFA) and batch-API (NNPFA and LPFA), use 3 nodes and collect samples over all available hardware threads (2 per core). Experiments with AVI (LVFA), run on a single node and employ a single thread. Each node is equipped with a 2.6 GHz AMD Rome 7H12 processor, has 128 CPU cores, and each core has 2 Gibibyte (GiB) of memory.