Use of Proximal Policy Optimization for the Joint Replenishment Problem*

Nathalie Vanvuchelen[a], Joren Gijsbrechts[a], Robert Boute[a,b]

[a] *Faculty of Economics and Business, KU Leuven, Belgium*
[b] *Vlerick Business School, Belgium*

Abstract

Deep reinforcement learning has been coined as a promising research avenue to solve sequential decision-making problems, especially if few is known about the optimal policy structure. We apply the proximal policy optimization algorithm to the intractable joint replenishment problem. We demonstrate how the algorithm approaches the optimal policy structure and outperforms two other heuristics. Its deployment in supply chain control towers can orchestrate and facilitate collaborative shipping in the Physical Internet.

Keywords: Collaborative Shipping, Physical Internet, Joint Replenishment Problem, Machine Learning, Deep Reinforcement Learning, Proximal Policy Optimization

1 Introduction

The recent digitization in transport provides new opportunities to improve the efficiency of current logistics networks. By sharing shipment data across companies, freight loads can be combined to increase truck fill rates. In the innovative concept of the Physical Internet, an open interconnected logistics network introduced by Montreuil (2011), companies collaborate by sharing freight and resources (Pan et al., 2017). This entails a transition from individually optimized supply chains towards a more holistic and collaborative integrated "network of networks". To facilitate smooth routing of freight across the network, digital control towers visualize shipments and support replenishment decisions based on real-time information and analytics.

As the foundation of the Physical Internet is built around multi-dimensional collaboration, it complements the emerging sharing economy in which goods and services are being shared and exchanged more easily (Beliën et al., 2017). Horizontal collaboration has been considered as effective practice for sustainable logistics and freight transport and it has gained increased attention in recent years (Pan et al., 2019). It is named as one of the solutions to effectively decarbonize freight transport (McKinnon, 2018; ALICE-ETP, 2019).

A prerequisite of collaborative shipping, either through bundling less-than-truckload (LTL) or backhauling full truckload (FTL) shipments, is that the replenishment cycles of the collaborating companies are synchronized, i.e., their replenishment occurs at the same time. This requires the implementation of a joint replenishment policy that takes into account the specifics of freight transport to stimulate joint shipments and avoid individual transports. The joint replenishment literature is rich and provides a range of heuristics.

*To appear in *Computers in Industry*

Most joint replenishment policies, however, ignore the limitation of the vehicle capacity and do not deal with the (under)utilization of the vehicles. We explore how machine learning can be used to develop joint replenishment policies that can be implemented in a control tower setting to facilitate collaborative shipping.

We implement a state-of-the-art machine learning algorithm to decide on the replenishments of a group of collaborating companies, i.e. how much and when to order or ship. We focus on periodic review joint replenishment policies that take the capacity of a full truck into account. The goal is to minimize (joint) transportation, holding and backorder costs. We deploy the proximal policy optimization (PPO) algorithm, a state-of-the-art optimization method in the domain of deep reinforcement learning (DRL). The PPO algorithm has been praised to capture some of the stability and convergence properties of trust region policy optimization algorithms, while being much simpler to implement and tune. Our numerical results confirm its stable and converging behaviour, while developing well-performing policies.

The implementation of machine learning algorithms, such as the one presented in this paper, can facilitate smart automation of freight shipments in today's digital era. It complements the algorithms developed in Creemers et al. (2017) to identify collaboration partners based on their geographical compatibility and the gain sharing methods, such as the ones discussed in Boute & Van Steendam (2018). As such this paper responds to various challenges faced by the current logistics industry, such as the transition towards the Physical Internet, more economically friendly eco-systems, and the digital transformation of logistics.

2 A review of capacitated joint replenishment policies

The joint replenishment problem (JRP) literature focuses on developing a replenishment policy for multiple items that minimizes the sum of inventory holding, backorder and ordering/transportation costs. The ordering costs typically include a major and a minor ordering cost, with the major ordering cost incurred every time an order is placed, independent of the number of items in the order, and the minor ordering cost charged for every item that is included in the order. The JRP can also be applied to a multi-company setting.

As the optimal policy becomes rapidly intractable for sizeable instances (Arkin et al., 1989), the JRP literature provides a range of well-performing (near-optimal) heuristics. The most applicable ones will be used as benchmark policies in our numerical experiment in Section 6. We focus on joint replenishment policies for stochastic demand, which can be classified into continuous and periodic review policies, depending on their review process. We refer the interested reader to Aksoy & Selcuk Erenguc (1988); Goyal & Satir (1989); Golany & Lev-er (1992); Khouja & Goyal (2008) and Bastos et al. (2017) for an extensive review, including joint replenishment policies for deterministic demand.

A well-known continuous review policy is the $(\mathtt{s}, \mathtt{c}, \mathtt{S})$ can-order-policy, introduced by Balintfy (1964). Under a can-order policy, an order is triggered every time the inventory position of an item i falls below its reorder point s_i. All other items of which the inventory position is below its can-order point c_i are also included in the order and their inventory positions are raised up to their order-up-to level S_i. Can-order policies have been used to understand the benefits of collaborative shipping in Padilla Tinoco et al. (2017).

Under a $(\mathtt{Q},\mathtt{S})$-policy, all items are replenished up to their order-up-to level S_i when the aggregate demand amounts to Q units (Renberg & Planche, 1967). Under a $\mathtt{Q}(\mathtt{s},\mathtt{S})$ policy (Viswanathan, 1997; Nielsen & Larsen, 2005), individual inventory positions are reviewed when the aggregate demand since the last order amounts to Q units, and all items of which the inventory position is lower than their reorder point s_i are ordered up to their order-up-to level S_i. A related policy is the $(\mathtt{Q},\mathtt{s},\mathtt{S})$ policy, proposed by Gürbüz et al. (2007), where a joint order is placed if the total demand observed since the last replenishment amounts to Q units *or* if the inventory position of an item i falls below its reorder point s_i. In that case, all items are ordered up to S_i.

Periodic review policies tend to be easier to implement and are therefore popular in industry. A classic periodic review policy is the $(\mathtt{R},\mathtt{T})$-policy (Atkins & Iyogun, 1988): inventory positions of all items are reviewed every T periods and are raised up to their order-up-to levels R_i. A variant of this policy allows the review period of item i to be a multiple k_iT of the base period T. Viswanathan (1997) suggests the periodic $\mathtt{P}(\mathtt{s},\mathtt{S})$ policy: every T periods, the inventory position of all items is reviewed and the inventory of all items with an inventory position below their reorder point s_i is raised up to their order-up-to-level S_i. The hybrid $(\mathtt{R},\mathtt{S},\mathtt{Q})$ policy combines continuous with periodic review: a joint order is triggered whenever the total demand since the last replenishment equals Q or if the time elapsed since the last replenishment equals R; in that case, the inventory position of all items is raised up to S_i (Yuksel Ozkaya et al., 2006).

None of the above policies explicitly take order costs into account in function of the order size (or they implicitly assume they are linear to the order quantities). In transportation, however, a transportation cost is charged per truck that is dispatched resulting in step-wise ordering costs. In what follows, we discuss continuous and periodic review policies that consider such a cost structure.

Cachon (2001), Tanrikulu et al. (2010) and Kiesmüller (2010) consider the aforementioned $(\mathtt{Q},\mathtt{S})$-policy, proposed by Renberg & Planche (1967), and assume a fixed cost per truck. Cachon (2001) compares the $(\mathtt{Q},\mathtt{S})$ policy with a periodic $(\mathtt{Q},\mathtt{S}\mid\mathtt{T})$ minimum order quantity policy. The $(\mathtt{Q},\mathtt{S}\mid\mathtt{T})$ policy works as follows: every T periods, the inventory position of item i is raised up to S_i. Orders are only shipped in FTLs, however, and the remaining quantity is shipped only if it exceeds Q units, with Q a decision variable. Tanrikulu et al. (2010) propose the $(\mathtt{s},\mathtt{Q})$-policy, where an order of Q units is triggered if the inventory position of an item i falls below its reorder point s_i. The Q units are then allocated such that the excess of the inventory position over the reorder point s_i is equalized for all items. In their numerical study, they consider numerical settings where Q is exogenous and equal to the capacity of a truck and settings where Q is a decision variable. The numerical experiments of Cachon (2001) and Tanrikulu et al. (2010) show that the truck utilization is higher when Q is a decision variable, i.e. Q increases towards the capacity of a truck, if the cost per truck or the lead time increases, or if the ratio of the truck capacity and the average total demand is close to one. Li & Schmidt (2019) extend the $(\mathtt{Q},\mathtt{S})$ policy to an $(\mathtt{U},\mathtt{S})$-policy, where an order is triggered if the cumulative demand *volume* exceeds the volume of a truck U.

Kiesmüller (2010) enforces FTL shipments in the $(\mathtt{Q},\mathtt{S})$ policy and the $(\mathtt{s},\mathtt{Q})$-policy, by setting Q equal to the capacity of a truck. Under a $(\mathtt{Q},\mathtt{S})$ policy, the order-up-to-levels S_i are optimized, whereas under an

(s, Q) policy, the Q units are allocated to each item based on an allocation procedure that maximizes the time until the next replenishment order is triggered (Miltenburg, 1985), so that each item has the same expected time until the subsequent replenishment. Büyükkaramikli et al. (2014) also consider the (s, Q) policy but additionally assume that the number of available trucks (fleet capacity) is limited.

Assuming step-wise order costs, Van Eijs (1994) extends the periodic (R, S) policy, where every R periods, the inventory position of an item i is initially raised up to S_i; these initial order quantities are increased to FTLs if that reduces the costs. Note that Van Eijs (1994) assumes different costs for LTL and FTL. Kiesmüller (2009) combines the ideas of Van Eijs (1994) and Cachon (2001) and proposes a periodic joint replenishment policy that allows initial order quantities to be increased *or* reduced at every review instant. To guarantee FTL shipments, dynamic individual order-up-to levels are used instead of constant order-up-to levels, based on the *aggregate* inventory position of the different items. To determine the individual order-up-to-levels, a myopic approach is used with a bisection method to find the optimal Lagrange multiplier of a Lagrange relaxation. Yang & Yano (2017) study a periodic (R, T)-policy that allows adjusting shipment quantities downward if the total amount of shipment quantities exceeds the truck capacity. In their numerical experiment, the truck capacity for the (R, T)-policy is set equal to the best Q for the (Q, S)-policy. Otero-Palencia et al. (2019) take into account transportation and warehouse constraints. They use a periodic cyclic (R, T) policy to synchronize replenishments where order quantities are restricted by a warehouse capacity.

We contribute to the literature by analyzing how DRL can be used to develop replenishment policies for the JRP with a cost structure that is relevant in a transportation context. The policy of Cachon (2001) and Kiesmüller (2009) both assume such cost structure and allow the shipment of multiple truckloads. We therefore choose to use them as benchmarks in our numerical study.

3 Deep reinforcement learning and its application in inventory management

Reinforcement learning (RL) is the branch of machine learning algorithms that focuses on sequential decision-making problems. An RL agent aims to learn a policy that maximizes future (discounted) rewards by interacting with an environment. For an extensive introduction to RL we refer to Sutton & Barto (2017). In contrast to supervised machine learning, where for instance a deep neural net is used to classify images and the function is approximated that maps pixels to the label of the image, RL algorithms contain many elements that can be approximated, such as the policies, the expected rewards, the model itself or a combination. Developing or applying an RL algorithm thus requires both art and science.

More traditional RL algorithms employ *simple* approximation architectures such as linear approximations. DRL uses a more complex, nonlinear approximator such as a neural net. Tesauro (1995) gained significant attention with his application of DRL on the well-known Backgammon game. Around the same time in the inventory literature, Van Roy et al. (1997) pioneers the use of DRL (also referred to as *Neuro-dynamic programming*), applied to a multi-echelon inventory model. Despite its good performance compared to a base-stock type policy, it requires a careful (manual) selection of the features of the state space in order to

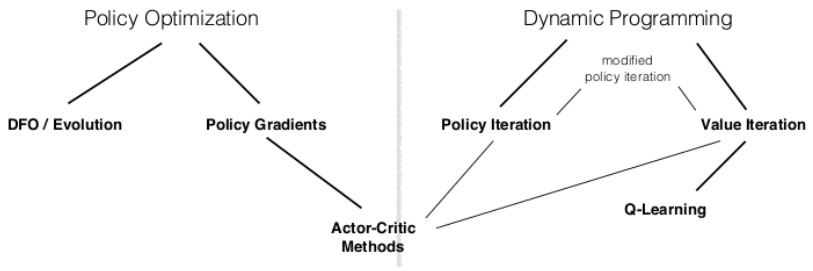


Figure 1: Overview of classes of reinforcement learning algorithms (Schulman, 2016).

make the algorithm work well. Interestingly, only a limited number of follow-up papers followed the first two decades after these seminal works. A renewed interest in DRL algorithms has recently sparked in the computer science field, followed by applications within inventory management. Notable works include the *AlphaGo* algorithm, exceeding human performance in the classic board game Go (Silver et al., 2017) and the Deep Q-learning algorithm excelling across a variety of Atari games (Mnih et al., 2013). In the wake of these developments, Oroojlooyjadid et al. (2017) use the Deep Q-learning algorithm to learn how to play the well-known beer supply chain game, while Gijsbrechts et al. (2019) test the performance of the actor-critic (A3C) algorithm as a general purpose technology on dual sourcing, lost sales and multi-echelon inventory problems. They coin DRL as a promising research avenue to solve intractable inventory control problems, but do highlight the extensive need for hyperparameter tuning. These observations motivate our choice for an algorithm that is less sensitive to the hyperparameters, yet with strong, empirically proven convergence properties. In what follows we introduce and discuss several classes of DRL algorithms and motivate why we opt for PPO in this paper.

Figure 1 provides an overview of the two frameworks that distinguish the RL algorithms: policy optimization and dynamic programming (DP). Policy optimization algorithms directly optimize a policy. They include evolutionary methods or derivative free optimization (DFO), which typically do not scale well (and out of scope for this article as we employ deep neural nets). Policy gradient algorithms rely on sampled decision sequences and apply gradient ascent to update the policy parameters. Pure policy gradient methods such as REINFORCE (Williams, 1988, 1992), sample decision sequences using Monte Carlo simulation and perform gradient ascent optimization to update the policy in the direction that maximizes the reward. Even though policy gradient methods have rather strong theoretical convergence guarantees, they suffer from *weak* sample complexity. They require an excessive (if not infinite) amount of samples to achieve good performance in many settings. This may cause policy gradient methods to be too slow to converge empirically, while the need to collect decision sequence samples may be impractical or expensive. This is relevant, for instance when training physical robots, as a low sample efficiency requires a lot of expensive robot training time. It is therefore desirable to have algorithms that converge fast to reasonably good policies.

A second class of algorithms leverages DP. They rely on Markov decision processes (MDP), which can be solved to optimality by solving the well-known optimality equations of Bellman (1954). This set of recursive

equations can be solved using value or policy iteration (Puterman, 1994). These DP methods are however restricted to sufficiently small problems (i.e., small demand support and/or short lead times). This led to the emergence of approximate dynamic programming (ADP) algorithms (Powell, 2007) that approximate the value or policy functions of MDPs. The class of RL algorithms that use the MDP framework falls within the umbrella of ADP. Deep Q-learning is an example of an RL algorithm that relies on a value function approximation. Q-learning methods benefit from lower sample complexity because they can reuse previous decision sequences more effectively than policy optimization algorithms. Nonetheless, we have a limited theoretical understanding about DRL algorithms that rely solely on value or policy function approximations. For instance, in contrast to its deterministic counterparts, there is no guaranteed theoretical performance to improve the policy in every iteration. This may eventually result in an arbitrarily bad policy. This is especially problematic when applying DRL algorithms in practice. Despite several empirical successes of DRL algorithms (Mnih et al., 2013; Oroojlooyjadid et al., 2017), there is no guarantee that the final policy performs well.

Actor-critic methods combine both policy optimization and DP frameworks by applying gradient ascent on the policy, developed by the actor, while relying on an estimation of the expected future reward, made by the critic. The A3C algorithm of Mnih et al. (2016) is such an algorithm, which has shown to perform well in an inventory context too (Gijsbrechts et al., 2019).

To circumvent the weak sample complexity of policy gradient methods while maintaining their convergence properties, Kakade (2002) uses conservative policy iteration in combination with the option to restart from past states. Their algorithm converges in a relatively small amount of steps to approximately optimal policies. Interestingly, they can provide an explicit bound by updating the policy more conservatively (as a mixture of the old and new policy). Schulman et al. (2015) develop the trust region policy optimization (TRPO) algorithm inspired by Kakade (2002). TRPO also improves monotonically in every training iteration by keeping the updates to the new policy within a specified range (measured using the KL-distance). TRPO performs generally well, but is computationally burdensome. It requires the solution to a constrained minimization problem in every iteration, which limits its application potential to larger problems. This inspired Schulman et al. (2017) to develop the PPO algorithm, which has been praised for its simplicity, good sample complexity and wall-time performance. In addition it is rather insensitive to its parameter tuning. Instead of mixing the old and new policy, PPO clips the distance between the new policy in comparison to the old policy. As such it mimics the trust region behaviour of the TRPO algorithm without the necessity to solve a constrained optimization problem in every training iteration. Moreover, it samples decision sequences similarly to policy gradient methods, but performs multiple updates on the same decision sequence. This makes PPO thus also more sample (or data) efficient.

Our choice for the PPO algorithm of Schulman et al. (2017) is thus motivated as follows: although PPO has to model both an actor and a critic, and therefore has a higher modelling complexity compared to Deep Q-learning and traditional policy gradient methods, it is widely praised in the computer science field to

capture the convergence properties of trust region methods, while being more sample efficient and easier to implement. Moreover, it has been empirically shown to be less sensitive towards its parameter tuning. That is why we have opted for this algorithm to tackle the intractable JRP problem.

4 Problem statement

We consider $n = 2$ shippers and a neutral orchestrator with full visibility on each shipper's demand and inventories, who is in charge of the coordination of their replenishments, for instance using a control tower. To ensure truck efficiency, shipments are only made in FTLs with a capacity of V units. The aggregated order quantity of both shippers in each period t is thus a multiple of an FTL of V units. Denote $q_{i,t}$ shipper i's order quantity and $M_t \in \mathbb{N}$ the number of truckloads shipped in period t, we have:

$$\sum_{i=1}^{n} q_{i,t} = M_t V, \tag{1}$$

indicating that either nothing ($M_t = 0$), or a multiple of an FTL is shipped. We assume that the replenishment lead time of each shipper is zero. The assumption of zero lead times allows a comparison with the optimal policy, and visualize how the policies developed by the PPO algorithm approximate the optimal policy structure.

The sequence of events is as follows: At the beginning of each review period t, order quantities $q_{i,t}$ are determined based on the inventory level at the end of the previous period $I_{i,t-1}$, i.e., before demand is observed. In case an order is placed, it is instantly replenished under the assumption of zero lead times. Then, demand $d_{i,t}$ is observed and satisfied. As per the conventional assumption in the JRP literature, we assume unmet demand is backordered. As orders are placed and replenished before demand is observed, the risk period is thus one period.

Costs are incurred at the end of each period. For every unit of shipper i, a holding cost h_i and backorder cost b_i is incurred per positive/negative unit of inventory on hand that is carried over to the next period. If an order is placed, a one-off minor order cost k_i is incurred if shipper i's items are included in the shipment (independent of the number of units of shipper i included in the order or the number of trucks dispatched). A major order cost K is incurred for every truck that is dispatched irrespective of the number of participating shippers. The costs in period t, denoted by c_t, are then given by:

$$c_t = \sum_{i=1}^{n} \left[h_i [I_{i,t}]^+ + b_i [-I_{i,t}]^+ + k_i \mathbb{1}_{\{q_{i,t}>0\}} \right] + M_t K. \tag{2}$$

The inventory decision-making of the collaborative joint replenishment can be formulated as an MDP. In an MDP, the process is in a state $s_t \in \mathcal{S}$, with $\mathcal{S}$ representing all states or the state space, and randomly transitions into a new state s_{t+1} after taking action $a_t \in \mathcal{A}_t$, with $\mathcal{A}_t$ representing all feasible actions that can be taken from state s_t. This transition provides a corresponding reward or cost $c_t(s_t, a_t)$. The transition from state s_t to s_{t+1} is determined by the transition probability $\mathbb{P}(s_{t+1} \mid s_t, a_t)$, i.e. the probability of ending up in state s_{t+1} if action a_t is taken in state s_t. In our joint replenishment problem, a decision is made every

period t on each item's order quantity $q_{i,t}$ based on the previous period end inventories $I_{i,t-1}$. The state s_t at time t is thus defined as:

$$s_t = \{I_{1,t-1}, ..., I_{n,t-1}\}, \tag{3}$$

and the actions $a_t = \{q_1, \ldots, q_n\}$ include each shipper's order quantities, which are restricted by Eq. (1). Eq. (2) specifies the costs $c_t(s_t, a_t)$ incurred in period t. After ordering $q_{i,t}$ and observing demand $d_{i,t}$, each element of the state transitions from s_t into s_{t+1} as per the inventory balance equations:

$$I_{i,t} = I_{i,t-1} + q_{i,t} - d_{i,t} \qquad \forall i \in \{1, ..., n\}. \tag{4}$$

The objective is to minimize the long-run cumulative costs, C_t. Assuming policy π_t is followed in state s_t, the cumulative costs from period t onwards are defined as

$$C_t^{\pi_t}(s_t) = \sum_{k=0}^{\infty} \gamma^k c_{t+k}(s_{t+k}, a_{t+k}), \tag{5}$$

where $\gamma \in (0, 1)$ is the discount factor. The discount factor determines the present value of future costs.

The expected value of these cumulative costs $C_t^{\pi_t}(s_t)$, corresponds to the value function $v^{\pi_t}(s_t)$, which indicates the value of being in state s_t following π_t:

$$v^{\pi_t}(s_t) = \sum_{k=0}^{\infty} \gamma^k \mathbb{E}^{\pi_t} c_{t+k}(s_{t+k}, a_{t+k}). \tag{6}$$

The value $v(s_t)$ is recursively related to the value of the next state, $v(s_{t+1})$. Obtaining the value functions of the optimal policy can be achieved by solving the well-known Bellman equations (Bellman, 1954). For each state $s_t \in \mathcal{S}$, their value functions satisfy:

$$v^\star(s_t) = \min_{a_t \in \mathcal{A}} \left\{ c_t(s_t, a_t) + \gamma \sum_{s_{t+1}} \mathbb{P}(s_{t+1} \mid s_t, a_t) v^\star(s_{t+1}) \right\} \quad \forall s_t \in \mathcal{S}. \tag{7}$$

For small-scale settings the Bellman equations can be solved to optimality by using DP algorithms. Unfortunately, DP suffers from the curse of dimensionality making them computationally intractable for sizeable problems. The curse of dimensionality is driven by the state space (of which the dimension grows in the number of collaborating shippers), the action space (which includes all possible ordering combinations) and the transition space (dependent on the number of possible demand realizations). DRL provides a way to circumvent this curse of dimensionality. In Section 3 we have motivated our choice for the PPO algorithm. We now elaborate in-depth on how to apply the PPO algorithm to find each shipper's order quantities $q_{i,t}$ that minimize the costs in Eq. (2).

5 Application of the proximal policy optimization algorithm

The PPO algorithm that we implement is an actor-critic algorithm that leverages elements from trust region policy optimization methods. A neural network is used to develop a policy $\pi(s_t; \theta)$, known as the actor network, and another neural net is used to estimate the value function $v^\pi(s_t; \phi)$ when following the policy of

the actor, which is the critic network. The values of the parameters θ and ϕ of these neural nets are set by *training* the algorithm, which we elaborate below. By clipping the updates from the old to the new policy, excessive differences in the policy are avoided, resulting in more stable training behaviour. Similarly to policy gradient methods, a buffer of training data is filled with combinations of states, actions and rewards, which is used to define the loss function. While traditional policy gradient algorithms only apply one update per sampled data point (i.e., they suffer from weaker sample complexity), PPO will perform multiple updates on the same training data. In what follows we elaborate in more detail on the neural network architecture and the actual training process.

We start by defining the approximation architecture. For the actor we use a fully connected neural network consisting of two hidden layers with width 128, both followed by a tanh activation function as used in Schulman et al. (2017). The tanh activation maps an input x to an output $\frac{e^x - e^{-x}}{e^x + e^{-x}}$. This results in output values in the range [-1,1], effectively creating non-linearity. The dimension of the first layer of the neural net equals the dimension of the state space $\mathcal{S}$. The dimension of the last layer, i.e., the output layer, equals the dimension of the action space $\mathcal{A}$ and uses a softmax activation function. For each state $s_t \in \mathcal{S}$ the actor network outputs the policy $\pi(s_t; \theta)$ which is the probability distribution over all actions. The parameters θ and ϕ of respectively the actor and the critic neural networks are not shared. The configuration of the critic network is identical to the actor network except for the output layer. The output layer of the critic has dimension one and outputs an estimation of the value function $v^{\pi}(s_t; \phi)$ which is defined as the expected future discounted costs when following policy π. The starting weights of the neural networks, parameterized by θ_0 for the actor and ϕ_0 for the critic, are initialized randomly. To obtain a well-performing policy and good estimates of the value function, the weights and biases of the neural nets are iteratively updated through applying adaptive mini-batch gradient descent (using the well-known Adam optimizer (Kingma & Ba, 2015)).

Each training iteration k, we collect a buffer of training data that is sampled following a decision sequence using the current policy of the actor π_{θ_k}. More specifically, we simulate m periods and store for each period the observed state, the action taken and the corresponding cost. For each visited state, we also store the action probability distribution under the previous policy $\pi_{\theta_{k-1}}$, which we obtain from the actor network with the model parameters from the previous iteration $k-1$. In our numerical experiment we employed a buffer of size $m = 256$ periods. Once the buffer is full, we update the old policy $\pi_{\theta_{k-1}}$. Hereafter, we predict the value functions $v^{\pi_k}(s_i; \phi) = \begin{bmatrix} v^{\pi_k}(s_1; \phi) & ... & v^{\pi_k}(s_m; \phi) \end{bmatrix}$ and calculate the future discounted costs $C = \begin{bmatrix} C_1 & ... & C_m \end{bmatrix}$. The cost C_i for sample point i that occurred at time t is defined as the sum of the discounted costs until the end of the episode and the discounted infinite horizon cost after the end of the episode, which we approximate by the value function in the last period of the episode:

$$C_i = \sum_{t'=t}^{m} \gamma^{t'-t} c_{t'} + \gamma^{m-t} v^{\pi_k}(s_m; \phi). \tag{8}$$

Based on the value function estimates and the future discounted cost calculations, we quantify the *advantage* of an action a when following policy π in state s. This gives an indication on how much worse/better

an action is than the expected cost provided by the critic. The intuition behind this is to update the actor network in a direction that favours better actions. To estimate the advantages $\hat{A}^{\pi_k} = \begin{bmatrix} \hat{A}_1^{\pi_k} & ... & \hat{A}_m^{\pi_k} \end{bmatrix}$, we use the generalized advantage estimation approach of Mnih et al. (2016). This approach makes use of a weighing parameter $\lambda \in [0, 1]$, which we set equal to one in our experiments. The advantage $\hat{A}_i^{\pi_k}$ for sample point i that occurred at time t is then calculated as follows:

$$\hat{A}_i^{\pi_k} = -v^{\pi_k}(s_t; \phi) + C_i. \tag{9}$$

To update the model parameters θ and ϕ of the actor and critic network, we apply mini-batch gradient descent. This method is between stochastic gradient descent (SGD) and batch gradient descent. Whereas SGD updates the model parameters for every sample point in the training buffer, batch gradient descent averages the gradients of all sample points and updates the model parameters based on this average gradient. Mini-batch gradient descent splits the training data into batches and calculates an average gradient for each batch. Hence, the number of updates to the networks is equal to the number of batches. We employ a batch size of 64 sample points, which means that we split our training buffer into four batches. In addition to that, we use our training data multiple times to update the model parameters of the neural nets. The number of times the entire decision sequence of m sample points is used to update the model parameters of the neural networks is defined as the number of epochs. We set the number of epochs equal to 10; in our numerical experiment in section 6.2 we show the impact of varying the number of epochs on the performance of the algorithm and its sample requirements. To reduce the variance between the updates and to avoid overfitting the neural nets to the sample data, we shuffle the training data in every epoch, resulting in different batch configurations and gradients. To conclude, we update the model parameters forty times and use each sample point ten times per training iteration. The latter improves sample complexity since sample points are used multiple times to update the model parameters, and averaging the gradient on smaller batches reduces the variance of the gradient.

For each update of the model parameters, we compute two loss functions. Since the parameters between the actor and the critic networks are not shared, the loss function of the critic consist of a value loss, whereas the loss function of the actor consists of a policy loss and an entropy loss. The value loss is used to update the difference between the future discounted cost C_i and the value function approximation $v^{\pi_k}(s_i; \phi)$ in every state s_i, known as the value function error of the critic. The value loss is defined as the mean squared error of all the states in the batch and is calculated as follows:

$$\text{Value loss} = \sum_{i \in \mathcal{P}} \left(v^{\pi_k}(s_i; \phi) - C_i \right)^2, \tag{10}$$

where $\mathcal{P}$ denotes the set of all the sample points in a batch.

The policy loss is used to improve the policy and can be formulated as follows:

$$\text{Policy loss} = \mathbb{E}_{\pi_k} \left[\sum_{i \in \mathcal{P}} \left[min(r_i(\theta_k) \hat{A}_i^{\pi_k}, clip(r_i(\theta_k), 1 - \epsilon, 1 + \epsilon) \hat{A}_i^{\pi_k}) \right] \right]. \tag{11}$$

Hyperparameter	Well-performing value
Number of hidden layers	2
Width of hidden layers	[128,128]
Loss clipping ϵ	0.2
Number of epochs	10
Discount factor γ	0.99
Buffer size	256
Batch size	64
Entropy factor β_E	10^{-5}
Learning rate	10^{-4}

Table 1: Hyperparameters used to obtain results outlined in Section 6.

The key distinguishing factor of the PPO algorithm compared to other DRL algorithms is that updates to the policy loss function are clipped using a parameter ϵ. This clipping works as follows. Let $r_i(\theta_k) = \pi_{\theta_k}(a_i|s_i)/\pi_{\theta_{k-1}}(a_i|s_i)$ be the probability ratio of the old policy $\pi_{\theta_{k-1}}$ and the new policy π_{θ_k}. The values of $r_i \hat{A}_i^{\pi_k}$ are then clipped to the range $[1-\epsilon, 1+\epsilon]$ to avoid that the update to the policy loss would be too large. It is empirically proven that this results in more stable training behaviour.

In addition to the policy loss, an entropy loss is added to stimulate the exploration of new actions. Entropy is a measurement of randomness and is defined as the logarithm of the probability mass function over actions taken: $\mathbb{P}(\cdot \mid s_i) \log \mathbb{P}(\cdot \mid s_i)$. If a policy has high entropy, it explores more. This can prevent the policy from convergence to bad performing local optima. The entropy factor β_E determines the importance of the entropy loss compared to the policy loss in the loss function of the actor. The entropy is minimized if all actions have the same probability, stimulating the algorithm to take new (unexplored) actions.

$$\text{Entropy loss} = \beta_E \sum_{i \in \mathcal{P}} \mathbb{P}(\cdot \mid s_i) \log \mathbb{P}(\cdot \mid s_i). \tag{12}$$

The actual training process that we have implemented, can be summarized as follows. We evaluate the performance of the current policy every 1000 training iterations, i.e., after 1000 buffers of 256 periods. The evaluation is carried out by simulating 10 sample paths of 100 000 periods, of which we exclude the first 10 000 warm-up periods. If the performance of the policy does not improve for 30 consecutive evaluation iterations, training is terminated. We occasionally experienced that we had to restart this training process, when PPO converged to a local optimum. Yet, we never had to restart more than three times to obtain our results. We thus did not have to perform time-consuming hyperparameter tuning, but relied on well-performing values found in seminal papers such as Schulman et al. (2017). We note that we did some experiments with different hyperparameter values, but these did not lead to better results. We summarize the hyperparameters that we have used in our implementation of the PPO algorithm in Table 1, together with the values we used to train the neural nets.

6 Results

We show the performance of the PPO algorithm to our joint replenishment problem in a numerical experiment. We first conduct a small-scale experiment with limited demand support and compare the policies developed by PPO with the optimal policies, found using DP. To solve the DP, we employ value iteration (Puterman, 1994) with discount factor $\gamma = 0.99$. We evaluate the cost performance, as well as the order quantities and the steady state distributions of the inventory levels. We then expand our study to a larger experiment in which we compare PPO against two heuristic policies. For these settings the DP is no longer tractable and we cannot compare against the optimal solution.

We evaluate the performance of our policies through simulation. For each policy, we did 10 simulation replications of 100 000 periods (and excluded a warm-up period of 10 000 periods from each sample). We report the average total cost per period together with a 95% confidence interval.

6.1 Benchmark policies

We use the periodic $(\mathtt{Q}, \mathtt{S} \mid \mathtt{T})$ minimum order quantity, proposed by Cachon (2001), and the periodic review dynamic order-up-to-policy (which we will abbreviate by $\mathtt{DYN-OUT}$), proposed by Kiesmüller (2009), as our benchmark heuristics. These heuristics are designed for our problem setting as they assume the same cost structure and allow the shipment of multiple truckloads per replenishment.

The $(\mathtt{Q}, \mathtt{S} \mid \mathtt{T})$ policy raises the inventory position of item i up to its order-up-to level S_i every T periods. Orders are only shipped in FTLs and the remaining order quantity is shipped if it exceeds Q units. S_i and Q are decision variables with $Q \in [1, V]$ and T is exogenous. The number of trucks required M_t is:

$$M_t = \left\lfloor \frac{\sum_{i=1}^{n} \left(S_i - I_{i,t}\right)}{V} \right\rfloor, \tag{13}$$

where $\lfloor x \rfloor$ denotes the largest integer that is less than or equal to x. The remaining quantity is shipped if:

$$\sum_{i=1}^{n} \left(S_i - I_{i,t}\right) - M_t \geq Q, \tag{14}$$

and the number of trucks that is dispatched then equals $M_t + 1$. If the remaining order quantity is smaller than Q, only M_t trucks are dispatched, in which case the total replenishment quantity VM_t needs to be allocated among the items. Cachon (2001) proposes a FIFO allocation policy where the items are ordered in the same sequence as demand was observed. As in our model demand is aggregated per period, a FIFO allocation is not useful. Instead, we use a proportional rule to allocate the replenishment quantity VM_t, so that the order quantity of item i is:

$$q_{i,t} = \frac{S_i - I_{i,t}}{\sum_{i=1}^{n} \left(S_i - I_{i,t}\right)} VM_t. \tag{15}$$

Under a $\mathtt{DYN-OUT}$ policy, replenishment decisions are periodically made based on the aggregate inventory position, $\sum_{i=1}^{n} I_{i,t}$. The order-up-to levels S_i are dynamically determined to fill truckloads of capacity V.

In a first step, the number of FTLs M_t is determined each period t that is required to raise the aggregate inventory position to an aggregate order-up-to level S_0:

$$M_t = \left[\frac{S_0 - \sum_{i=1}^{n} I_{i,t}}{V} \right], \tag{16}$$

where $[x]$ denotes the closest integer to x and the aggregate order-up-to level S_0 is the sum of the optimal individual order-up-to-levels S_i^*:

$$S_0 = \sum_{i=1}^{n} S_i^*, \tag{17}$$

and each S_i^* is determined through the newsvendor solution:

$$F_{d_i}(S_i^*) = \frac{b_i}{b_i + h_i}, \tag{18}$$

with F_{d_i} the cumulative distribution function of demand d_i during one period (as we assume zero lead times). The total shipment quantity, VM_t, is then allocated among all items using an order-up-to policy. The order quantity $q_{i,t}$ for item i in period t is then:

$$q_{i,t} = \max\{0, \hat{S}_{i,t} - I_{i,t}\}, \tag{19}$$

with $\hat{S}_{i,t}$ the order-up-to level of item i in period t, determined by a myopic approach and Lagrange relaxation:

$$F_{d_i}(\hat{S}_{i,t}^*) = \frac{b_i + \lambda}{b_i + h_i}, \tag{20}$$

where λ is a Lagrange multiplier. To guarantee FTL shipments the values of $\hat{S}_{i,t}$ have to satisfy:

$$\sum_{i=1}^{n} (\hat{S}_i - I_{i,t}) = M_t V. \tag{21}$$

A bisection method is used to determine the optimal value of the Lagrange multiplier λ.

6.2 Small-scale numerical experiment

We study 16 different settings where the DP is tractable, listed in Table 2, in which the truck capacity is $V = 6$ units. In these settings, it is optimal for the $(\mathtt{Q}, \mathtt{S} \mid \mathtt{T})$ policy to order in FTLs, which allows a fair comparison between the different heuristics. The settings can be divided into four groups, with four possible configurations of the shipper's holding cost within each group: low-low ($h_1 = h_2 = 1$), low-high ($h_1 = 1, h_2 = 5$), high-low ($h_1 = 5, h_2 = 1$) and high-high ($h_1 = h_2 = 5$). The minor ordering cost of shipper 1 is different in the second and fourth group ($k_1 = 40$ versus $k_1 = 10$). Settings 1-8 and 9-16 also differ in their demand: $d_1 = U[0, 5]$ and $d_2 = U[0, 3]$ versus $d_1 = U[0, 6]$ and $d_2 = U[0, 2]$.

Figure 2 shows the optimality gaps of the PPO algorithm, the $(\mathtt{Q}, \mathtt{S} \mid \mathtt{T})$ and the $\mathtt{DYN-OUT}$ heuristics in each of the 16 settings. It shows that PPO develops close-to-optimal policies and significantly outperforms our benchmarks. These results are promising and urge the question as to why they perform better.

Figure 3 visualizes the order quantities of both shippers in function of their inventory levels, as well as the steady state probabilities of their inventory levels. For each policy, represented by a column in Figure 3,

	Parameters												
Setting	V	n	h_1	h_2	b_1	b_2	k_1	k_2	K	d_1	d_2	l_1	l_2
1	6	2	1	1	19	19	10	10	75	U[0,5]	U[0,3]	0	0
2	6	2	1	**5**	19	**95**	10	10	75	U[0,5]	U[0,3]	0	0
3	6	2	**5**	1	**95**	19	10	10	75	U[0,5]	U[0,3]	0	0
4	6	2	**5**	**5**	**95**	**95**	10	10	75	U[0,5]	U[0,3]	0	0
5	6	2	1	1	19	19	**40**	10	75	U[0,5]	U[0,3]	0	0
6	6	2	1	5	19	95	**40**	10	75	U[0,5]	U[0,3]	0	0
7	6	2	5	1	95	19	**40**	10	75	U[0,5]	U[0,3]	0	0
8	6	2	5	5	95	95	**40**	10	75	U[0,5]	U[0,3]	0	0
9	6	2	1	1	19	19	10	10	75	**U[0,6]**	**U[0,2]**	0	0
10	6	2	1	5	19	95	10	10	75	**U[0,6]**	**U[0,2]**	0	0
11	6	2	5	1	95	19	10	10	75	**U[0,6]**	**U[0,2]**	0	0
12	6	2	5	5	95	95	10	10	75	**U[0,6]**	**U[0,2]**	0	0
13	6	2	1	1	19	19	40	10	75	**U[0,6]**	**U[0,2]**	0	0
14	6	2	1	5	19	95	40	10	75	**U[0,6]**	**U[0,2]**	0	0
15	6	2	5	1	95	19	40	10	75	**U[0,6]**	**U[0,2]**	0	0
16	6	2	5	5	95	95	40	10	75	**U[0,6]**	**U[0,2]**	0	0

Table 2: Settings for the small-scale experiment

the matrix in the first row represents the order quantities of shipper 1 under that policy. Similarly, the order quantities for shipper 2 are represented in the second row. From these order quantities, it can be verified that the total shipped quantity $q_1 + q_2$ equals a multiple of a truckload, $M_t V$, with M_t the number of trucks and the truckload $V = 6$ units. The matrix in the third row visualizes the steady state probabilities of the inventory levels of both shippers.

Suppose for instance that the system is in state $(I_1, I_2) = (5, 0)$. Under the optimal policy, only shipper 2 orders $q_2 = 6$ units, while $q_1 = 0$. This is equivalent to one FTL. The same orders are placed under the policy developed by PPO. The $(\mathtt{Q}, \mathtt{S} \mid \mathtt{T})$ and $\mathtt{DYN-OUT}$ policies, in contrast, both order $q_1 = 2$ and $q_2 = 4$ units. From the steady-state probabilities, it can be seen that this state is regularly visited.

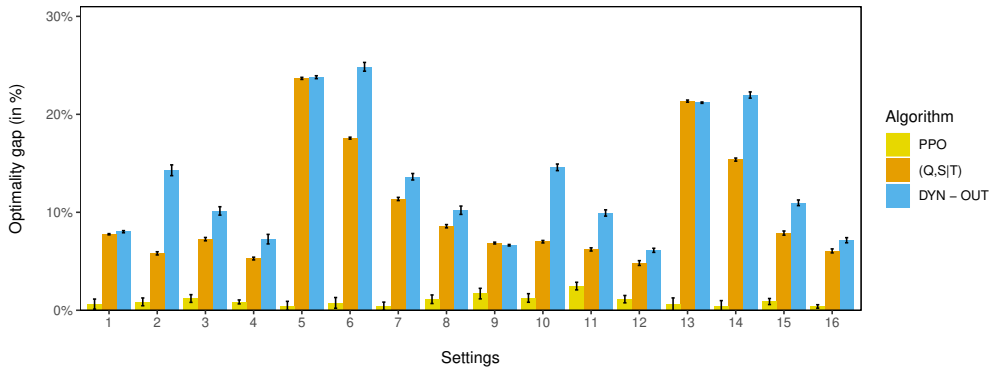


Figure 2: The PPO algorithm performs close to optimal, whereas the $(\mathtt{Q}, \mathtt{S} \mid \mathtt{T})$ and the $\mathtt{DYN-OUT}$ heuristics perform between $4 - 25\%$ of the optimal policy in our settings. The optimal cost performance is taken as a baseline and is compared to the average costs found by PPO and the heuristics. A 95% confidence interval is provided for the simulation results.

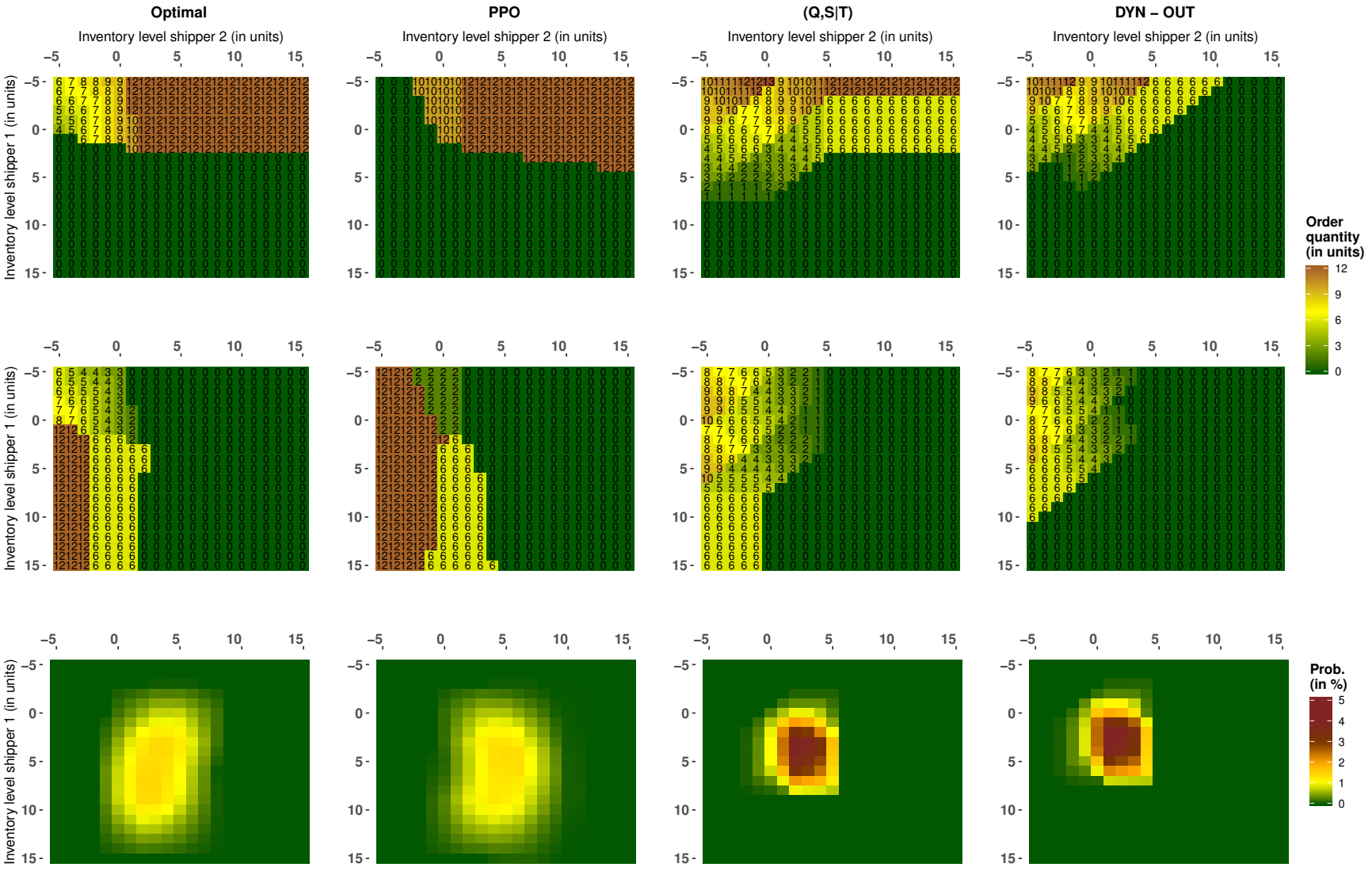


Figure 3: Visualisation of the replenishment policy and steady state inventory probabilities for $h_1 = h_2 = 1$, $k_1 = 40$, $k_2 = 10$, $K = 75$, $d_1 = U[0, 5]$ and $d_2 = U[0, 3]$. Top: order quantities for shipper 1 depending on the inventory levels of shippers 1 and 2. Middle: order quantities for shipper 2. Bottom: steady state inventory probabilities. The PPO policy (second column) approaches the optimal policy (first column), whereas the the $(\mathtt{Q}, \mathtt{S} \mid \mathtt{T})$ (third column) and the $\mathtt{DYN} - \mathtt{OUT}$ (fourth column) show different behavior.

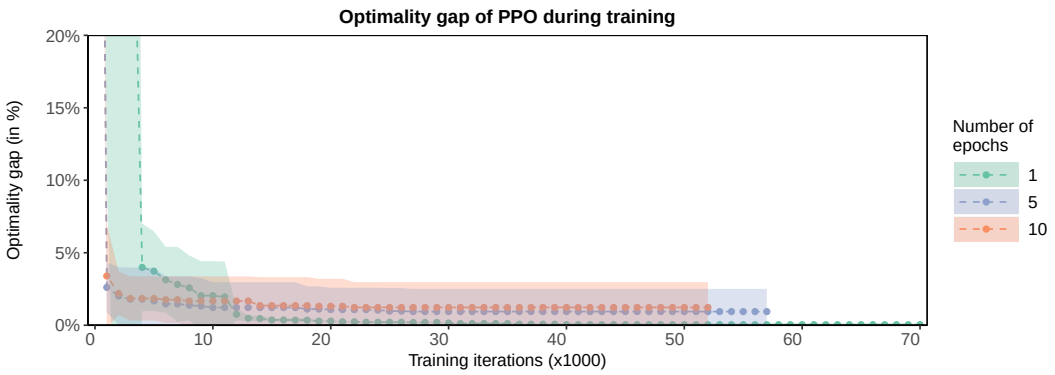


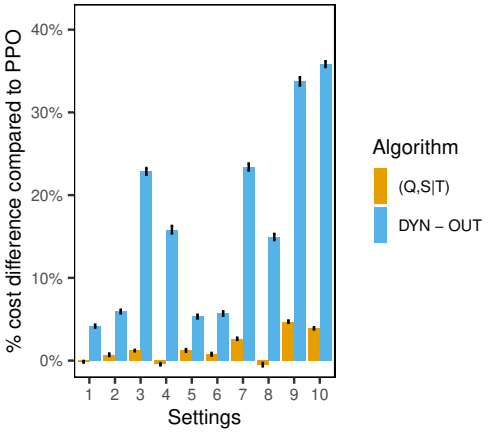
Figure 4: Visualisation of the average best solution (dotted lines) and its standard deviation (shaded area) during training for $h_1 = h_2 = 1$, $k_1 = 40$, $k_2 = 10$, $K = 75$, $d_1 = U[0, 5]$ and $d_2 = U[0, 3]$. The average best solution and the standard deviation increase whereas the number of training iterations decreases with the number of epochs. A small number of epochs results in stable convergence (small standard deviation), but requires a lot of training samples. Increasing the number of epochs reduces the need for training samples at the risk of overfitting to the training data and getting stuck in local optima (high standard deviation).

The colour code reveals how the PPO algorithm (second column) develops policies that resemble the optimal policy (first column), but is different from the policy developed by the $(\mathtt{Q}, \mathtt{S} \mid \mathtt{T})$ heuristic (third column) and much different from the $\mathtt{DYN} - \mathtt{OUT}$ policy (fourth column). Note that the regions of interest are the (frequently) visited inventory states. Finally, note that compared to PPO and the optimal policy, the steady-state inventory distributions under the $(\mathtt{Q}, \mathtt{S} \mid \mathtt{T})$ and the $\mathtt{DYN} - \mathtt{OUT}$ heuristics are more centered and shifted left-upwards. These policies tend to keep lower stock levels and replenish more often than the optimal policy and the policy found by PPO.

We also carried out an experiment to analyze the sample efficiency of the PPO algorithm. As already mentioned in Section 5, traditional policy gradient methods only perform one update per sampled data point whereas PPO is able to perform multiple updates on the same sample point. How often a sample point is used to update the model parameters is determined by the number of epochs. Therefore, we studied three configurations for the number of epochs: 1, 5 and 10. For each configuration, the algorithm was run three times and we kept track of the best solutions found during training. In Figure 4, we show the impact of the number of epochs on the performance of the PPO algorithm and its sample requirements. We plot the average best solutions found during training (represented by the dotted lines) and their corresponding standard deviation (represented by the shaded areas). As can be seen from Figure 4, the algorithm converges slower to the best policy if every sample point is used only once. However, the standard deviation is very small indicating that for each run of the algorithm the solution found is similar. Interestingly, the average best solution and its standard deviation increases with the number of epochs whereas the number of training iterations decreases with the number of epochs. The former is due to overfitting on the sampled data causing the algorithm to get stuck in a local optimum every so many runs. In conclusion, a trade-off can be made between policy performance and sample efficiency.

	Parameters												
Setting	V	n	h_1	h_2	b_1	b_2	k_1	k_2	K	d_1	d_2	l_1	l_2
1	33	2	1	1	19	19	10	10	400	U[15,25]	U[5,15]	0	0
2	33	2	1	**5**	19	**95**	10	10	400	U[15,25]	U[5,15]	0	0
3	33	2	**5**	1	**95**	19	10	10	400	U[15,25]	U[5,15]	0	0
4	33	2	**5**	**5**	**95**	**95**	10	10	400	U[15,25]	U[5,15]	0	0
5	33	2	1	1	19	19	**40**	10	400	U[15,25]	U[5,15]	0	0
6	33	2	1	5	19	95	**40**	10	400	U[15,25]	U[5,15]	0	0
7	33	2	5	1	95	19	**40**	10	400	U[15,25]	U[5,15]	0	0
8	33	2	5	5	95	95	**40**	10	400	U[15,25]	U[5,15]	0	0
9	33	2	**7**	1	**133**	19	**40**	10	400	U[15,25]	U[5,15]	0	0
10	33	2	**8**	1	**152**	19	**40**	10	400	U[15,25]	U[5,15]	0	0

(a)



(b)

Figure 5: For the settings listed in (a), (b) shows that the policy found by PPO performs comparable to the $(\mathtt{Q}, \mathtt{S} \mid \mathtt{T})$ heuristic for settings in which the cost structure of shippers is similar but outperforms both the $(\mathtt{Q}, \mathtt{S} \mid \mathtt{T})$ and the $\mathtt{DYN} - \mathtt{OUT}$ heuristic in settings in which their cost structure is different.

6.3 Large-scale numerical experiment

We carried out the same analysis for an experiment with larger demand support. As the DP can no longer be solved due to its exploding state space, we only compare the PPO results against the $(\mathtt{Q}, \mathtt{S} \mid \mathtt{T})$ and the $\mathtt{DYN} - \mathtt{OUT}$ heuristics. We study eight different settings, summarized in Figure 5a, in which the truck capacity is $V = 33$ (as a fully loaded truck usually carries 33 pallets). The configuration and the assumptions of these settings are similar to those in Table 2.

Figure 5b presents the relative cost difference between the policies found by PPO and the $(\mathtt{Q}, \mathtt{S} \mid \mathtt{T})$ and the $\mathtt{DYN} - \mathtt{OUT}$ heuristics. We observe that the policy found by PPO performs similarly to the $(\mathtt{Q}, \mathtt{S} \mid \mathtt{T})$ heuristic in case the cost structure of the shippers is similar (setting 1, 4, 5 and 8), but outperforms the $(\mathtt{Q}, \mathtt{S} \mid \mathtt{T})$ heuristic if the shippers have a different cost structure (setting 2, 3, 6, 7, 9 and 10). These results are in line with the findings of Pantumsinchai (1992) and Viswanathan (1997), who find that the $(\mathtt{Q}, \mathtt{S})$-policy performs well if items have similar or identical demand and cost characteristics, the number of items is small and the major setup cost is high compared to the minor order cost. This is similar to our settings, where the major order cost is high and shippers have similar cost parameters in certain settings. We observe that the performance of the $(\mathtt{Q}, \mathtt{S} \mid \mathtt{T})$ policy is worse, compared to the policy found by PPO, when the shippers become less similar.

7 Conclusion

In this paper we use proximal policy optimization to solve the joint replenishment problem with full truckload shipments. Proximal policy optimization is a deep reinforcement learning algorithm, that performs well without extensive hyperparameter tuning. We show in a small-scale experiment with limited demand support how proximal policy optimization develops policies that approximate the optimal policy structure

and outperforms the periodic review minimum order quantity and dynamic order-up-to heuristics. In a larger experiment, where the optimal policy is unknown, we find that proximal policy optimization performs similar to the minimum order quantity heuristic when the shippers (or items) have a similar cost structure, but outperforms both heuristics when the cost structure is dissimilar. The implementation of these smart learning algorithms in digital control towers can support the transition towards more collaborative shipping in a Physical Internet.

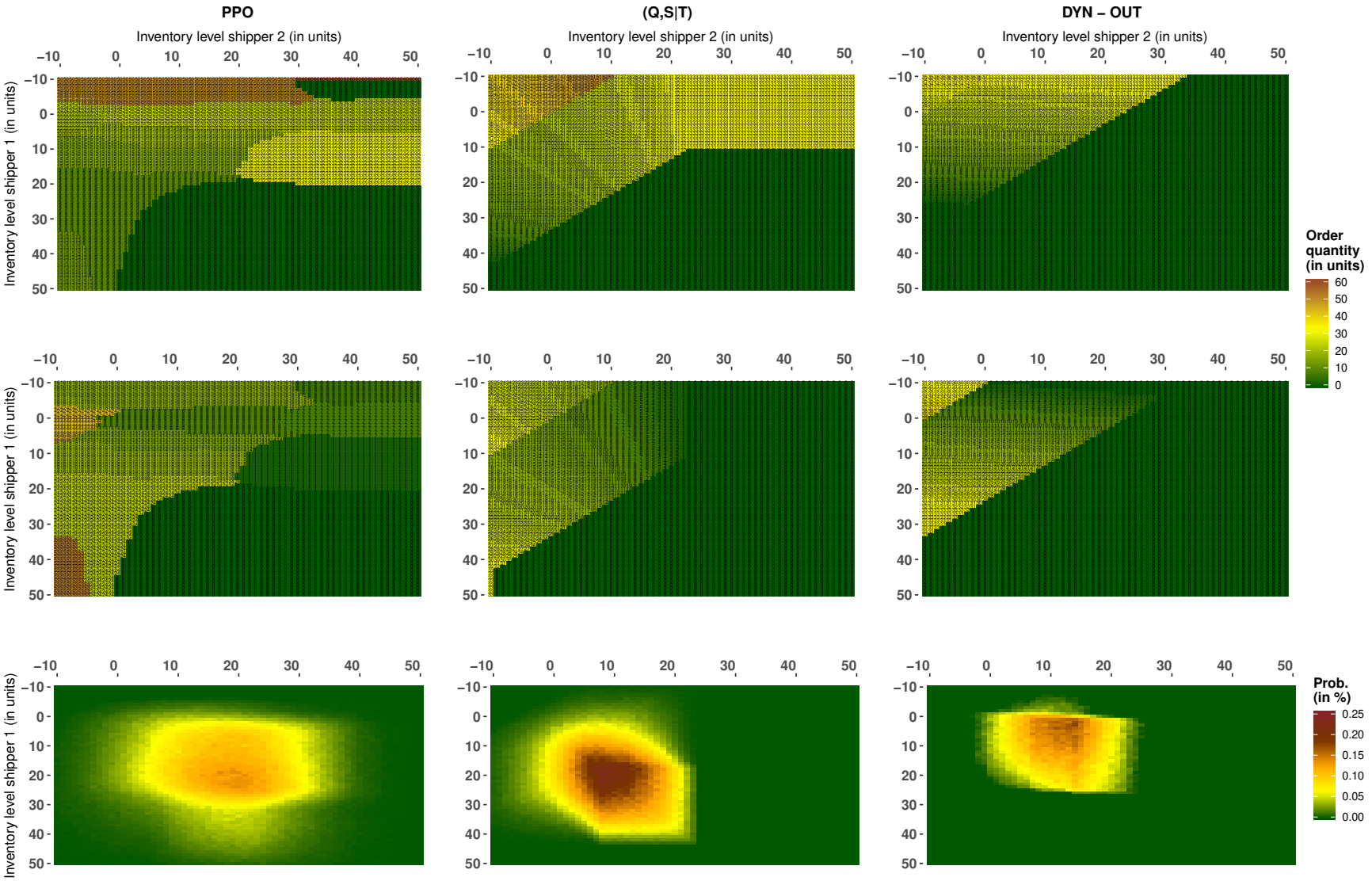


Figure 6: Visualisation of the replenishment policy prescribed by the PPO algorithm, the $(\mathtt{Q}, \mathtt{S} \mid \mathtt{T})$ and the $\mathtt{DYN} - \mathtt{OUT}$ heuristic for $h_1 = 7$, $h_2 = 1$, $k_1 = 40$, $k_2 = 10$, $K = 400$, $d_1 = U[15, 25]$ and $d_2 = U[5, 15]$. Top: order quantities for shipper 1 depending on the inventory levels of shipper 1 and 2. Middle: order quantities for shipper 2. Bottom: steady state probabilities of the inventory levels.

References

Aksoy, Y., & Selcuk Erenguc, S. (1988). Multi-item inventory models with co-ordinated replenishments: a survey. *International Journal of Operations & Production Management*, *8*, 63–73.

ALICE-ETP (2019). A framework and process for the development of a roadmap towards zero emissions logistics 2050. URL: `http://www.etp-logistics.eu/wp-content/uploads/2019/12/Alice-Zero-Emissions-Logistics-2050-Roadmap-WEB.pdf`.

Arkin, E., Joneja, D., & Roundy, R. (1989). Computational complexity of uncapacitated multi-echelon production planning problems. *Operations Research Letters*, *8*, 61–66.

Atkins, D., & Iyogun, P. (1988). Periodic versus "can-order" policies for coordinated multi-item inventory systems. *Management Science*, *34*, 791–796.

Balintfy, J. L. (1964). On a basic class of multi-item inventory problems. *Management science*, *10*, 287–297.

Bastos, L. d. S. L., Mendes, M. L., Nunes, D. R. d. L., Melo, A. C. S., & Carneiro, M. P. (2017). A systematic literature review on the joint replenishment problem solutions: 2006-2015. *Production*, *27*.

Beliën, J., Boute, R., Creemers, S., De Bruecker, P., Gijsbrechts, J., Padilla Tinoco, V., & Verheyen, W. (2017). Collaborative shipping: Logistics in the sharing economy. *ORMS Today*, *44*, 20–23.

Bellman, R. (1954). The Theory of Dynamic Programming. *Bulletin of the Amer Math Soc*, *60*, 503–515.

Boute, R., & Van Steendam, T. (2018). A better way to share the gains of collaborative shipping. *Supply Chain Management Review*, *22*, 36–41.

Büyükkaramikli, N. C., Gürler, Ü., & Alp, O. (2014). Coordinated logistics: joint replenishment with capacitated transportation for a supply chain. *Production and Operations Management*, *23*, 110–126.

Cachon, G. (2001). Managing a retailer's shelf space, inventory, and transportation. *Manufacturing & Service Operations Management*, *3*, 211–229.

Creemers, S., Woumans, G., Boute, R., & Belien, J. (2017). Tri-vizor uses an efficient algorithm to identify collaborative shipping opportunities. *Interfaces*, *47*, 244–259.

Gijsbrechts, J., Boute, R. N., Van Mieghem, J. A., & Zhang, D. (2019). Can Deep Reinforcement Learning Improve Inventory Management? Performance and Implementation of Dual Sourcing-Mode Problems. *SSRN Electronic Journal*, . doi:`10.2139/ssrn.3302881`.

Golany, B., & Lev-er, A. (1992). Comparative analysis of multi-item joint replenishment inventory models. *International Journal of Production Research*, *30*, 1791–1801.

Goyal, S. K., & Satir, A. T. (1989). Joint replenishment inventory control: deterministic and stochastic models. *European journal of operational research*, *38*, 2–13.

Gürbüz, M., Moinzadeh, K., & Zhou, Y.-P. (2007). Coordinated replenishment strategies in inventory/distribution systems. *Management Science*, *53*, 293–307.

Kakade, J., Sham; Langford (2002). Approximately optimal approximate reinforcement learning. *Proceedings of the Nineteenth International Conference on Machine Learning*, *2*, 267–274.

Khouja, M., & Goyal, S. (2008). A review of the joint replenishment problem literature: 1989–2005. *European Journal of Operational Research*, *186*, 1–16.

Kiesmüller, G. P. (2009). A multi-item periodic replenishment policy with full truckloads. *International Journal of Production Economics*, *118*, 275–281.

Kiesmüller, G. P. (2010). Multi-item inventory control with full truckloads: A comparison of aggregate and individual order triggering. *European Journal of Operational Research*, *200*, 54–62.

Kingma, D. P., & Ba, J. (2015). Adam: A Method for Stochastic Optimization. In *ICLR 2015*. `arXiv:1412.6980`.

Li, L., & Schmidt, C. P. (2019). A stochastic joint replenishment problem with dissimilar items. *Decision Sciences*, .

McKinnon, A. (2018). *Decarbonizing Logistics: Distributing Goods in a Low Carbon World*. (1st ed.). Kogan Page.

Miltenburg, G. J. (1985). Allocating A Replenishment Order Among A Family of Items. *IIE Transactions*, *17*, 261–267.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. `arXiv:1602.01783`.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. `arXiv:1312.5602`.

Montreuil, B. (2011). Toward a physical internet: meeting the global logistics sustainability grand challenge. *Logistics Research*, *3*, 71–87.

Nielsen, C., & Larsen, C. (2005). An analytical study of the Q(s,S) policy applied to the joint replenishment problem. *European Journal of Operational Research*, *163*, 721 – 732.

Oroojlooyjadid, A., Nazari, M., Snyder, L., & Takáč, M. (2017). A Deep Q-Network for the Beer Game with Partial Information. `arXiv:1708.05924`.

Otero-Palencia, C., Amaya-Mier, R., & Yie-Pinedo, R. (2019). A stochastic joint replenishment problem considering transportation and warehouse constraints with gainsharing by shapley value allocation. *International Journal of Production Research*, *57*, 3036–3059.

Padilla Tinoco, S. V., Creemers, S., & Boute, R. N. (2017). Collaborative shipping under different cost-sharing agreements. *European Journal of Operational Research*, *263*, 827–837.

Pan, S., Ballot, E., Huang, G. Q., & Montreuil, B. (2017). Physical internet and interconnected logistics services: research and applications. *International Journal of Production Research*, *55*, 2603–2609.

Pan, S., Trentesaux, D., Ballot, E., & Huang, G. Q. (2019). Horizontal collaborative transport: survey of solutions and practical implementation issues. *International Journal of Production Research*, *57*, 5340–5361.

Pantumsinchai, P. (1992). A comparison of three joint ordering inventory policies. *Decision Sciences*, *23*, 111–127.

Powell, W. B. (2007). *Approximate dynamic programming: solving the curses of dimensionality*. Wiley-Interscience.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. (1st ed.). New York, NY, USA: John Wiley & Sons, Inc.

Renberg, B., & Planche, R. (1967). Un modèle pour la gestion simultanèe des n articles d'un stock. *Revue française d'informatique et de recherche opérationnelle*, *6*, 47–59.

Schulman, J. (2016). *Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs*. Ph.D. thesis EECS Department, University of California, Berkeley. URL: `http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-217.html`.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., & Abbeel, P. (2015). Trust Region Policy Optimization. `arXiv:1502.05477`.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. `arXiv:1707.06347`.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A. et al. (2017). Mastering the game of go without human knowledge. *Nature*, *550*, 354.

Sutton, R. S., & Barto, A. G. (2017). *Reinforcement Learning: An Introduction*. MIT Press Cambridge.

Tanrikulu, M., Şen, A., & Alp, O. (2010). A joint replenishment policy with individual control and constant size orders. *International Journal of Production Research*, *48*, 4253–4271.

Tesauro, G. (1995). Temporal difference learning and td-gammon. *Communications of the ACM*, *38*, 58–68.

Van Eijs, M. (1994). Multi-item inventory systems with joint ordering and transportation decisions. *International Journal of Production Economics*, *35*, 285–292.

Van Roy, B., Bertsekas, D. P., Lee, Y., & Tsitsikis, J. N. (1997). A Neuro-Dynamic Programming Approach to Retailer Inventory Management. *Proceedings of the IEEE Conference on Decision and Control*, .

Viswanathan, S. (1997). Periodic review (s, S) policies for joint replenishment inventory systems. *Management Science*, *43*, 1447.

Williams, R. J. (1988). *Toward a theory of reinforcement-learning connectionist systems*. Technical Report Northeastern University, College of Computer Science.

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

Yang, K.-C., & Yano, C. A. (2017). Multi-product (R, T) inventory policies with short-shipping: Enabling fixed order intervals and well-utilized vehicles under random demand. *IISE Transactions*, *49*, 209–222.

Yuksel Ozkaya, B., Gürler, ü., & Berk, E. (2006). The stochastic joint replenishment problem: A new policy, analysis, and insights. *Naval Research Logistics*, *53*, 525 – 546.