Multi-echelon inventory optimization using deep reinforcement learning

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

ORIGINAL PAPER

Multi-echelon inventory optimization using deep reinforcement learning

Kevin Geevers[1] · **Lotte van Hezewijk[1,2]** · **Martijn R. K. Mes[3]**

Abstract

This paper studies the applicability of a deep reinforcement learning approach to three different multi-echelon inventory systems, with the objective of minimizing the holding and backorder costs. First, we conduct an extensive literature review to map the current applications of reinforcement learning in multi-echelon inventory systems. Next, we apply our deep reinforcement learning method to three cases with different network structures (linear, divergent, and general structures). The linear and divergent cases are derived from literature, whereas the general case is based on a real-life manufacturer. We apply the proximal policy optimization (PPO) algorithm, with a continuous action space, and show that it consistently outperforms the benchmark solution. It achieves an average improvement of 16.4% for the linear case, 11.3% for the divergent case, and 6.6% for the general case. We explain the limitations of our approach and propose avenues for future research.

Keywords Deep reinforcement learning · Proximal policy optimization · Multi-echelon · Inventory control · Backorders

✉ Kevin Geevers
   kevin.geevers@ortec.com

   Lotte van Hezewijk
   l.v.hezewijk@tue.nl

   Martijn R. K. Mes
   m.r.k.mes@utwente.nl

[1] ORTEC B.V., Houtsingel 5, 2719 EA Zoetermeer, The Netherlands

[2] Industrial Engineering & Innovation Sciences, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

[3] Faculty of Behavioural, Management and Social Sciences, University of Twente, P.O.Box 217, 7500 AE Enschede, The Netherlands

🖄 Springer

1 Introduction

Inventory is usually kept to respond to fluctuations in demand and supply. With more inventory, a higher service level can be achieved but the inventory costs also increase. To find the right balance between the service level and inventory costs, inventory management is needed. Inventory usually accounts for 20–60% of the total assets of manufacturing firms (Unyimadu and Anyibuofu 2014). Therefore, inventory management policies prove critical in determining the profit of such firms (Arnold et al. 2008). Most companies are already using inventory policies to manage their inventories. With these policies, they determine how much to order at a certain point in time, as well as how to maintain appropriate stock levels to avoid shortages. These policies often focus on a single location and only use local information, which results in individually optimized local inventories and do not benefit the supply chain as a whole. The reason for not expanding the scope of the policies is the lack of sufficient data and the growing complexity of the policies. Due to recent IT developments, it has become easier to exchange information between stocking points, resulting in more data availability. This contributed to an increasing interest in inventory optimization over multiple locations, also called Multi-echelon inventory optimization (MEIO). Studies show that the coordination among inventory policies, i.e. a centralized decision-making policy, can reduce the bullwhip effect (Giannoccaro and Pontrandolfo 2002).

Despite the promising results, a lot of companies are still optimizing individual locations (Jiang and Sheng 2009). Mathematical models for inventory management can quickly become too complex and too time-consuming to solve. To prevent this, the models usually heavily rely on assumptions and simplifications (Jiang and Sheng 2009; Dogru et al. 2013; Jain and Raghavan 2009), for example regarding lead times, demand patterns, and network structures, which makes it harder to relate the models to real-world problems and to be put into practice. Tunc et al. (2011) demonstrate that certain assumptions (in their case stationary demand patterns) that do not hold in real-life, can be very costly when using the corresponding models in real-life applications. Another way of reducing the complexity and computation time is the use of (meta)heuristics. Unfortunately, these heuristic policies are typically problem-dependent and still rely on assumptions, which may limit their use in different settings (Gijsbrechts et al. 2022). Although mathematical models and heuristics can deliver good results for their specific setting, we are especially interested in a method that can be used for various supply chains, without requiring major modifications. For such a generic approach, reinforcement learning (RL) is a promising method that may cope with this complexity (Topan et al. 2020). RL is developed to solve sequential decision making problems in dynamic environments (Sutton and Barto 2018). In sequential decision making, a series of decisions are to be made in interaction with a dynamic environment to maximize overall rewards (Shin and Lee 2019). This makes RL an interesting method for inventory management. Due to recent advances in deep learning (Goodfellow et al. 2016), reinforcement learning is often combined with neural networks. This combination, called deep reinforcement learning (DRL)

(Mnih et al. 2013), shows even more potential to solve complex inventory problems. Inspired by a real case at a manufacturing company, which we refer to anonymously as the CardBoard Company, we recognize the need for an approach to solve MEIO for general inventory systems with stochastic demand, which is more complex than previously has been considered in the literature.

This paper applies DRL to three multi-echelon inventory systems. For each of these systems, we take into account the holding and backorder costs. We model the problem as a Markov decision process (MDP) and solve it using the DRL approach proximal policy optimization (PPO). The contribution of this paper is threefold. First, we solve these inventory problems by proposing order quantities that depend on various features of the inventory system. To determine the order quantities, we use a continuous action space. Most other research on applications of DRL in supply chain management use a discrete action space. The discrete action space has limitations in terms of problem size, whereas the continuous action space is more scalable and can therefore be used to cope with real-life problems. Second, one of the cases we consider is the inventory problem of a general inventory system (besides the simpler linear and divergent systems) with backorders. While the general inventory system is relevant for practice, it has received limited attention in research, and to the best of our knowledge, DRL has not been applied to a general inventory system with this many stock points before. Third, we showcase a method that can be used without major modifications for various supply chains by solving three different cases. Two of the cases are from literature, while one case is based on our case study at the CardBoard Company. This flexibility allows the application of this general learning algorithm without having to rely on considerable domain knowledge or limiting assumptions.

The remainder of this paper is structured as follows. In Sect. 2, we perform a literature study on the application of DRL in multi-echelon inventory systems. Section 3 introduces the three different cases on which we will apply our DRL method. The PPO algorithm and the setup of the experiments are introduced in Sect. 4. In Sect. 5, the results of our DRL method are presented. We end with conclusions and discussion in Sect. 6.

2 Literature review

In this section, we first elaborate on studies concerning multi-echelon inventory management (Sect. 2.1). Second, we zoom in on the advances in RL research in Sect. 2.2. Finally, we look at the literature that uses RL as an approach for solving inventory management problems in Sect. 2.3.

2.1 Multi-echelon inventory management

The term multi-echelon is used for supply chains where an item moves through more than one stage (i.e., location) before reaching the final customer (Ganeshan 1999; Rau et al. 2003). Because of the multiple locations, managing inventory in

a multi-echelon supply chain is considerably more difficult than managing it in a single-echelon one (Gumus et al. 2010). As a result, this research area attracted the attention of many scientists. During the last decade, the research focus shifted more to integrated control of the supply chain, mainly due to advanced information technology (Kalchschmidt et al. 2003; Rau et al. 2003; Gumus and Guneri 2009). According to Gumus and Guneri (2007), mathematical models are mostly used, often in combination with simulation, followed by heuristics and Markov Decision Processes.

The MEIO problem is characterized by the network structure of the supply chain. The simplest structure is a linear network, where each stock point is supplied by one upstream location, and is supplying one downstream location. A more complicated structure is the divergent network, where one stock point can supply multiple downstream locations. The most complicated structure is the general structure, which contains at least one stock point that can be supplied by more than one upstream location, while also containing at least one stock point that can supply multiple downstream locations. Figure 1 provides a visualization of these network structures, where we consider warehouses w (stock points) and retailers r. Retailers are stock points where demand is observed and can be supplied by one or multiple warehouses.

The aim of exact mathematical models is to solve the inventory management problem to proven optimality. Lambrecht et al. (1984) shows that the multi-echelon stochastic problem can be modeled as a Markov decision process (MDP). An MDP consists of stages where we transition from one state to the next, satisfying the Markov property (where the next state is dependent only on the current state and the chosen action). Several researchers model the MEIO problem as an MDP, as multiple convenient methods for solving MDPs exist. However, these authors typically focus on relatively simple structures of the supply chain, for example, the linear supply chain Lambrecht, Luyten and Vander Eecken 1985; Chen and Song 2001;
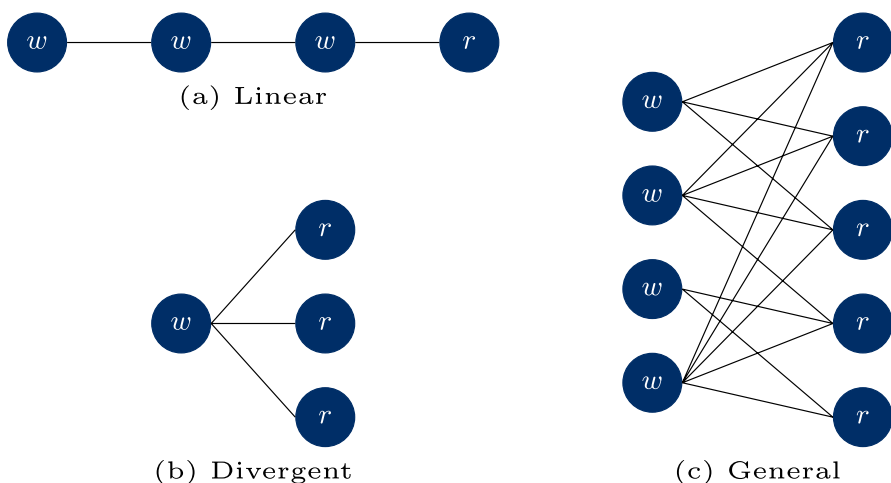

Fig. 1 Three types of network structures for MEIO problems

Vercraene and Gayon 2013; Iida 2001) and the divergent supply chain (Chen et al. 2002; Minner et al. 2003). Yet, determining optimal policies is intractable, even for simple network structures, due to the well-known curses of dimensionality (De Kok et al. 2018; Bellman 1957). As a result, most mathematical models use extensive restrictions and assumptions, such as deterministic demand and constant lead times. Also, mathematical models are usually complex and case-specific.

There are solution approaches that aim to overcome the aforementioned curses of dimensionality. In Sect. 2.2, we discuss approximate methods for solving the MDP, the focus of this paper. Another approach, Robust Optimization, is applied by Bertsimas and Thiele (2006) and Ben-Tal et al. (2009) to the MEIO problem, where the demand uncertainty is captured in uncertainty sets. Because Robust Optimization methods typically yield conservative solutions, measures must be taken to avoid the focus on possible worst-case realizations of the uncertainty. Bertsimas and Thiele (2006) and Ben-Tal et al. (2009) show that this method yields promising results, and is less affected by the curses of dimensionality.

Due to the intractability of mathematical models, heuristics are also often used as a solution approach in the optimization of multi-echelon inventory management. Heuristics are used to find a near-optimal solution in a short time span and are often a set of general rules, which can, therefore, be used on various problems. Frequently, a heuristic is used to optimize the parameters of the order-up-to policy (i.e., base-stock policy). This policy has been the most commonly adopted approach in inventory optimization (Nahmias and Smith 1993, 1994). For multi-echelon inventory systems, the base-stock policy, though not necessarily optimal, has the advantage of being simple to implement and to perform well (Rao et al. 2000; Rong et al. 2017).

2.2 Reinforcement learning

MDPs are typically solved using dynamic programming (DP) (Bellman 1957). Because of the high computation costs of DP, an approximate method is often required to solve MDPs. We consider two approximate approaches: (1) Approximate dynamic programming (ADP) and (2) RL. ADP is a modeling framework based on an MDP model (Mes and Rivera 2017), whereas RL does not necessarily assume a perfect model (Sutton and Barto 2018). ADP is an umbrella term for a broad spectrum of methods to approximate the optimal solution of an MDP, just like RL. It typically combines optimization with simulation. ADP and RL are two closely related paradigms for solving sequential decision-making under uncertainty, but in the remainder of this paper, we will use the umbrella term RL.

RL is about learning a policy to maximize a reward. In RL, unlike most forms of machine learning, the learner is not explicitly told which action to perform in each state (Sutton and Barto 2018). At each time step, the agent observes the current state s_t and chooses an action a_t to take. The agent selects the action by trial-and-error (exploration) and based on its knowledge of the environment (exploitation) (Giannoccaro and Pontrandolfo 2002). After taking the action, the agent is given a reward r_{t+1} and evolves in the new state s_{t+1}. Using this observed information about the state transition and reward, the agent updates its knowledge of the environment

and selects the next action. To maximize the reward, the trade-off between exploration and exploitation is important. The agent has to explore new state-action pairs in order to find the value of the corresponding reward. Full exploration ensures that all actions are taken in all states to avoid ending up in a local optimum. However, this is often too time consuming, so most of the methods start with full exploration and gradually increase the chance for exploitation.

In RL, tables could be used that map all action values to the states. Nevertheless, this table grows prohibitively large when solving large, realistic problems. Furthermore, the learnings of some states that are similar to others (i.e. having values for the state variables that are close to each other) cannot easily be generalized with this representation. To address this problem, often approximations for policies or value functions are used. With approximations, only the parameters of the function need to be stored, rather than the whole mapping table. There are several ways to use value or policy function approximation in RL, but a method that is gaining popularity is the use of deep learning (DL). Deep learning means that the learning method uses one or multiple neural networks to estimate the value or policy function. In a recent paper, Mnih et al. (2015) applied deep learning as a value function approximator and gained impressive results. Deep learning was combined with RL, because they have a common goal of creating general-purpose artificial intelligence systems that can interact with and learn from their environment (Arulkumaran et al. 2017). While RL provides a general-purpose framework for decision-making, deep learning provides a general-purpose framework for representation learning. With the use of minimal domain knowledge and a given objective, it can learn the best representation directly from raw inputs (Sutton and Barto 2018). The neural networks are updated with gradient descent, where the objective is to minimize the loss function.

We can roughly distinguish between value-based and policy-based RL. In value-based RL methods, the values of states and actions are approximated with a function, which could still result in a tremendous computational load. In this case, the loss indicates how well the estimates of the values match the observed values. Examples of value-based methods are Q-Learning (where the action with the maximum estimate is used in the updates) and Temporal Difference Learning (where the action that is taken is used in the updates). In essence, in solving the MDP, the optimal policy is the desired end result. With policy-based methods, the policy is directly parameterized and approximated, rather than learning 'indirectly' through the value functions. In this case, the loss indicates how well the policy is performing. Usually, some type of baseline is used in policy-gradient methods that reduces the variance in learning. Value- and policy-based methods can also be combined, for example in the A3C algorithm where the value estimates (i.e., the critic) are used as a baseline for the policy (i.e., the actor).

Schulman et al. (2017) developed the proximal policy algorithm (PPO), which is more scalable, sample-efficient, and requires less parameter tuning than other DRL algorithms. It is a policy-gradient algorithm that can be used in combination with a critic. The key of the algorithm is that large updates to the policy are avoided by clipping the loss function that is used to perform stochastic gradient ascent updates. This ensures that the weights of the neural network are updated in a subtle way, resulting in more stable training behavior. It has been shown empirically that the

performance of this PPO algorithm is less sensitive to the chosen hyper-parameters compared to other DRL algorithms. Furthermore, PPO is applicable to a continuous action space. If there are a large number of potential actions in a range of continuous values, it is possible to learn the parameters of the probability distribution of the action (continuous action space), rather than the probabilities of every individual action (discrete action space). This enhances the scalability of the algorithm to larger problem instances.

2.3 Reinforcement learning in multi-Echelon inventory management

Quite some research on RL or ADP in MEIO has already been done. An overview is given in Table 1. Within this overview, we list per paper the considered network structure and how demand that is not satisfied is treated. Additionally, the objective used in solving the problem is of relevance. Some papers focus on cost (and/ or profit), while others focus on minimizing the difference between the realized and target service levels. The problem horizon is also a consideration. In a finite horizon, RL is trained by simulating a given time period, e.g., a number of days or weeks. The outcome of such a finite horizon approach is a time-dependent policy, which suggests an action for each possible state in each time period of the horizon. In an infinite horizon, the simulation runs for an arbitrary time period, and the outcome will be a stationary policy. In the last column of Table 1, we show the approach that is used by the authors. Note that some entries are incomplete in case the research did not clearly specify these characteristics.

As we can see in Table 1, there are already a large number of papers available considering RL in MEIO. The papers differ substantially in their inventory systems and approaches. When we compare the different network structures of the papers, we can see that a divergent network is mostly assumed, followed by a linear supply chain. Only one paper considers a convergent structure and four papers describe a general network structure, which is the network structure of our case study at the CardBoard Company. In two papers where this structure is considered, the authors recommend addressing the scalability challenge to solve larger problems (Çimen and Kirkbride 2013, 2017).

In the case of unsatisfied demand, both lost sales and backorders are almost equally considered in literature, whereas two papers consider a hybrid approach. Reaching the target service level is the goal in eight papers, and most of the papers focus on the costs by either minimizing the costs or maximizing the profit. The algorithms that are used vary, but Q-learning is the most frequently used algorithm, followed by the actor-critic (AC) algorithm including variations of this, such as A3C. While RL and ADP are both often used in multi-echelon inventory management, DRL is considered nine times. After reviewing these papers, we conclude that, while reinforcement learning is used because of its promise to solve large problems, many papers still use extensive assumptions and simplifications. As we see in the table, the problems that are solved in literature are still quite different from the general network setting of the CardBoard Company. Harsha et al. (2021) consider a general network structure and a dual-sourcing structure, which are both quite different from

Table 1 Usage of reinforcement learning in multi-echelon inventory management

Paper	Network structure	Unsatisfied demand	Objective	Horizon	Approach
Carlos et al. (2008)		Backorders	Minimize costs	Finite	RL–Q learning
Chaharsooghi et al. (2008)	Linear	Backorders	Minimize costs	Finite	RL–Q learning
Çimen and Kirkbride (2013)	General	Lost sales	Minimize costs	Infinite	ADP-TD(λ)
Çimen and Kirkbride (2017)	General	Lost sales	Minimize costs	Infinite	ADP- Q learning
Dittrich and Fohlmeister (2020)	Linear	Backorders	Minimize costs	Finite	DRL-DQN
Elson and Brintrup (2021)	Linear	Guaranteed service	Minimize costs	Finite	DRL-A2C
Geng et al. (2010)	Divergent	Lost sales	Minimize costs	Finite	ADP-Value based
Giannoccaro and Pontrandolfo (2002)	Linear	Backorders	Maximize profit	Infinite	RL–Q learning
Gijsbrechts et al. (2022)	Divergent	Hybrid	Minimize costs	Infinite	DRL-A3C
Harsha et al. (2021)	Divergent General	Backorders Lost sales	Maximize profit	Infinite	DRL-Value based
Jiang and Sheng (2009)	Divergent	Lost sales	Target service level	Infinite	RL-Value based
Kim et al. (2005)	Divergent	Lost sales	Target service level	Infinite	RL-Value based
Kim et al. (2008)	Linear	Backorders	Minimize costs	Infinite	RL-Value based
Kim et al. (2010)	Linear		Target service level	Infinite	RL-Value based
Kunnumkal and Topaloglu (2011)	Divergent	Backorders	Minimize costs	Finite	ADP
Kwak et al. (2009)	Linear	Backorders	Minimize costs	Infinite	RL-Value based
Kwon et al. (2008)	Divergent	Lost sales	Target service level	Infinite	RL-TD(λ)
Li et al. (2008)	Divergent	Lost sales	Target service level	Infinite	RL-Value based
Oroojlooyjadid et al. (2017)	Linear	Backorders	Minimize costs	Infinite	DRL-DQN
Pirhooshyaran and Snyder (2020)	Linear Divergent General	Backorders	Minimize costs	Finite	DRL-Value based
Peng et al. (2019)	Divergent	Backorders	Maximize profit		DRL-Value based
Rao et al. (2003)	Divergent	Lost sales	Maximize profit	Infinite	RL-Q learning
Ravulapati et al. (2004)	Divergent	Lost sales	Maximize profit	Infinite	RL-Q learning

Table 1 (continued)

Paper	Network structure	Unsatisfied demand	Objective	Horizon	Approach
Saitoh and Utani (2013)	Linear	Backorders	Minimize costs	Infinite	RL–Q learning
Shervais and Shannon (2001)	Divergent		Minimize costs	Finite	ADP-Value based
Shervais et al. (2003)	Divergent		Minimize costs	Finite	ADP-Value based; neural networks
Sui et al. (2010)	Divergent	Lost sales	Maximize profit	Infinite	RL–Q learning
Vanvuchelen et al. (2020)	Divergent	Backorders	Minimize costs	Infinite	DRL-PPO
Van Roy et al. (1997)	Divergent	Hybrid	Minimize costs	Infinite	ADP-TD(λ); neural networks
Van Tongeren et al. (2007)	Linear	Backorders	Minimize costs	Finite	RL–Q learning
Woerner et al. (2015)	Convergent	Backorders	Minimize costs	Infinite	ADP-Value based
Xu et al. (2009)	Linear	Lost sales	Minimize costs 4Target service level	Infinite	RL-Value based
Yang and Zhang (2015)	Linear		Target service level	Infinite	RL–Q learning
Zarandi et al. (2013)	Divergent	Backorders	Target service level	Infinite	RL-Value based
Zhang et al. (2013)	Divergent	Lost sales	Target service level	Infinite	DRL-Value based
This paper	**Linear Divergent General**	**Backorders**	**Minimize costs**	**Infinite**	**DRL-PPO**

our general network structure. In their general network structure, each retailer stock point can only be supplied by one warehouse, thus resembling a divergent network. In their dual sourcing structure, the network is fully connected, i.e., every retailer stock point (3 in total) is connected to each of the two warehouses. In our general network, the number of retailers (5) and warehouses (4) is larger, and the network is not fully connected, i.e., not all retailers can order at all warehouses. Pirhooshyaran and Snyder (2020) focus on a finite horizon policy and specify the order up to levels per node, whereas we focus on finding a steady-state policy and the replenishment quantities rather than order up to levels. Furthermore, both studies use a value-based DRL method to solve the problems, whereas we focus on the policy-based method PPO. This method has also been applied by Vanvuchelen et al. (2020), but in that case, a divergent joint-replenishment network is considered using a discrete action space. This method will scale poorly to the size of the general network considered in this paper, so we use a continuous action space within the PPO algorithm.

We contribute to the existing literature by considering multiple network structures, including the general structure, and applying a novel method within this field. By applying PPO with a continuous action space on multiple network structures, we study the flexibility of this method in solving such problems.

3 Problem statement

In this section, we introduce the multi-echelon inventory optimization problem with a general network structure. In this problem, we consider two types of stock points: warehouses $w \in W$ and retailers $r \in R$. A summary of the problem notation is given in Table 2.

We model the problem as a Markov decision process (MDP), where all decisions are made centrally by a single decision maker. In each time period t, the process is in state s_t, and the decision maker may choose an action a_t that is available in that state. The state (1) should contain enough information such that the problem has the Markov property, i.e., the future (e.g., the decision a_t and the expected future rewards) depends only on the present state and does not depend on past history. For each of the three inventory cases, we define the state as follows:

$$
\begin{aligned}
s_t = & \left[ti, tb, I_{w,t}^{end}, I_{r,t}^{end}, BO_{r,w,t}, BO_{r,t}, IT_{w,t,t'}, IT_{w,r,t,\tilde{t}} \right] \\
& \forall w \in W, r \in R, t' \in 1, \ldots, \lambda_w, \tilde{t} \in 1, \ldots, \lambda_{w,r}
\end{aligned}
\tag{1}
$$

In (1), ti represents the total inventory in the system, and tb defines the total number of backorders. Note that we do not consider the time period as a state variable, meaning that we are looking for a stationary (or infinite horizon) policy that is valid at any time period.

In our problem, the state consists of all the necessary information for the decision maker to take an action. In all problem structures, the action (2) to take is the order quantity for the upstream location. Therefore, we define the action space as an order for every edge of the network:

Table 2 Notation

Indices	
w	Warehouses, $w \in W$
r	Retailers, $r \in R$
t	Time period index, $t = 1, 2, \ldots, T$
Parameters	
$h_w, h_r \geq 0$	Holding cost per period for warehouse w, retailer r
$b_w, b_r \geq 0$	Backorder cost per period for warehouse w, retailer r
$\lambda_w \geq 0$	Lead time for transport to warehouse w
$\lambda_{w,r} \geq 0$	Lead time for transport from warehouse w to retailer r
$d_{r,t} \geq 0$	Demand at retailer r in period t, for $r \in R$
Variables	
$I_{w,t}^{begin} \in \mathbb{Z}$	Inventory level of warehouse w in period t after receiving incoming shipments
$I_{r,t}^{begin} \in \mathbb{Z}$	Inventory level of retailer r in period t after receiving incoming shipments from warehouses
$I_{w,t}^{end} \in \mathbb{Z}$	Inventory level of warehouse w in period t after sending outgoing shipments to retailers
$I_{r,t}^{end} \in \mathbb{Z}$	Inventory level of retailer r in period t after fulfilling customer demand
$BO_{r,w,t} \in \mathbb{Z}$	Backorders of warehouse w to retailer r in period t
$BO_{r,t} \in \mathbb{Z}$	Backorders of retailer r to customers in period t
$IT_{w,r,t,t'} \in \mathbb{Z}$	Number of items in transit from warehouse w to retailer r shipped in period t to arrive in period t'
$IT_{w,t,t'} \in \mathbb{Z}$	Number of items in transit to warehouse w shipped in period t to arrive in period t'
$O_{r,w,t} \in \mathbb{Z}$	Number of items ordered from retailer r to warehouse w in period t
$O_{w,t} \in \mathbb{Z}$	Number of items ordered from warehouse w to their supplier in period t

$$a_t = \left[O_{w,t}, O_{r,w,t} \right] \qquad \forall w \in W, r \in R \tag{2}$$

After taking the action, we have a transition to a new state s_{t+1}. The probability of moving to the new state s_{t+1} is influenced by the chosen action, and given by the state transition function $P(s_{t+1} \mid s_t, a_t)$. The transition function is presented by the sequence of events (Table 3), where the randomness lies in the uncertainty of customer demand $d_{r,t}$ and lead time $\lambda_{w,r}$ between stock points.

In every time step t, five events happen. At the beginning of each period, the shipments of the upstream stock points ($IT_{w,t',t}, IT_{w,r,t',t}$) are received. Then, the supplier of the warehouse sends outgoing shipments to the warehouse ($IT_{w,t,t+\lambda_w}$) with (possibly stochastic) lead time λ_w. Thereafter, the warehouses fulfill the orders $O_{r,w,t-1}$ from the retailers with on-hand inventory if possible ($IT_{w,r,t,t+\lambda_{w,r}}$). Transport between two stock points w and r requires a (possibly stochastic) lead time $\lambda_{w,r}$. If there is not enough inventory for all downstream orders, the retailers are ranked in ascending order of their inventory minus their backorders, and the one with the lowest inventory minus backorders is satisfied first, until there is no inventory left. When there is no inventory left, the (partial) order will be

Table 3 Sequence of events describing the transition dynamics of the MDP

Event 1	Receive incoming shipments

$I_{w,t}^{begin} = I_{w,t-1}^{end} + \sum_{t' \leq t} IT_{w,t',t}, \qquad (\forall w \in W)$

$I_{r,t}^{begin} = I_{r,t-1}^{end} + \sum_{w \in W, t' \leq t} [IT_{w,r,t',t}] \qquad (\forall r \in R)$

Event 2	Warehouse supplier observes warehouse orders from $t-1$ and sends outgoing shipment $\qquad (\forall w \in W)$

$IT_{w,t,t+\lambda_w} = IT_{w,t,t+\lambda_w} + O_{w,t-1}$

Event 3	Warehouses observe retailer orders and send outgoing shipments to retailers $(\forall w \in W)$

$I'_{w,t} = I_{w,t}^{begin}$

if $\sum_{r \in R} [O_{r,w,t-1} + BO_{r,w,t}] \leq I'_{w,t}$ **then**

$\qquad IT_{w,r,t,t+\lambda_{w,r}} = O_{r,w,t-1} + BO_{r,w,t} \qquad (\forall r \in R)$

else

$\qquad$ **for** r in ascending order of $I_{r,t} - \sum_{w \in W} BO_{r,w,t}$

$\qquad \qquad IT_{w,r,t,t+\lambda_{w,r}} = \max(\min(I'_{w,t}, O_{r,w,t-1} + BO_{r,w,t}), 0)$

$\qquad \qquad BO_{r,w,t+1} = O_{r,w,t-1} + BO_{r,w,t} - IT_{w,r,t,t+\lambda_{w,r}}$

$\qquad \qquad I'_{w,t} = I'_{w,t} - IT_{w,r,t,t+\lambda_{w,r}}$

$I_{w,t}^{end} = I'_{w,t}$

Event 4	Warehouses place upstream orders $O_{w,t}$, Retailers place upstream orders $O_{r,w,t}$

Event 5	Retailers fulfill customer demand $\qquad (\forall r \in R)$

$I_{r,t}^{end} = \max(I_{r,t}^{begin} - BO_{r,t} - d_{r,t}, 0)$

$BO_{r,t+1} = \max(BO_{r,t} + d_{r,t} - I_{r,t}^{begin}, 0)$

put in backorder ($BO_{r,w,t}$). After sending the outgoing shipments, upstream orders are placed ($O_{w,t}, O_{r,w,t}$). Finally, the customer demand is fulfilled at the retailers, and demand that was not satisfied is backordered ($BO_{r,t}$). This means that the demand of the customers $d_{r,t}$ is not known before the replenishment orders are placed. Note that in the linear case, as shown in Fig. 1, all stock points except the last one are considered warehouses. With respect to the sequence of events in the linear case, event 2 concerns only the most upstream warehouse. In event 3, there also exist warehouse to warehouse shipments and backorders (i.e., $IT_{w,w',t,t+\lambda_{w,w'}}, BO_{w,w',t}$ where $w, w' \in W$). In event 4, there are also warehouse to warehouse orders (i.e., $O_{w,w',t}$ where $w, w' \in W$).

The transition from s_t to s_{t+1} due to action a_t results in a reward $R(s_t, a_t, s_{t+1})$. In this general network, we minimize the costs, consisting of holding costs h_w, h_r and backorder costs b_w, b_r, for the warehouses w and retailers r, respectively. The action a_t directly impacts the inventory levels and backorders in the system. This results in the following cost function that we aim to minimize:

$$c_t(s_t, a_t) = \sum_{w \in W} \left[h_w \cdot I_{w,t} + b_w \cdot \sum_{r \in R} BO_{r,w,t} \right] + \sum_{r \in R} \left[h_r \cdot I_{r,t} + b_r \cdot BO_{r,t} \right] \qquad (3)$$

The objective of the MDP is to find a policy π^* that minimizes the discounted infinite sum of costs, where a policy is a mapping of states to actions. Thus, the

objective can be expressed as shown in (4), where γ is the discount factor balancing the impact of future and present rewards and a_t^π represents the action at time t prescribed by policy π. Note that we aim to minimize the long-term average costs. This means that in practice, we choose T to be high enough to reflect the infinite horizon costs.

$$\pi^* = \arg\min_{\pi \in \Pi} \mathbb{E}\left[\sum_{t=0}^\infty \gamma^t \cdot c_t\left(s_t, a_t^\pi\right)\right] \tag{4}$$

In the next section, we elaborate on how the PPO algorithm can be used to solve this problem.

4 Proximal policy optimization implementation

The PPO algorithm (Schulman et al. 2017) can be used to solve MDPs in an approximate manner. The policy π is prescribed by a neural network (actor network), and by interacting with the environment (i.e., a simulation of the problem dynamics, sampling demand, etc.) the parameters of the neural network are updated to improve the policy. In Sect. 4.1 we elaborate on the standard steps taken in the PPO algorithm to learn a good policy, and in Sect. 4.2 we describe the hyperparameters that are used in the algorithm. In the sections that follow, we specify how the state and action spaces and reward function were transformed to make them applicable for use in the PPO algorithm. Although we use a standard implementation of PPO, the hyperparameters, and the state, action, and reward space transformations are design choices.

4.1 Algorithm steps

We implement the actor-critic structure of the PPO algorithm as described by Schulman et al. (2017). Within this algorithm, two neural network approximators are used: one for the policy function and one for the value function. We call them the actor network ($\pi(s_t;\theta)$) and the critic network ($v^\pi(s_t;\phi)$), which do not share weights. The actor network gives the current policy, meaning that, given the input state, we get the action that is taken in that state. The critic network outputs the estimated value of being in that state. The network weights are initialized randomly, and then the training procedure starts. The random initialization encourages exploration at the beginning of training. The actor network results in a stochastic policy, as the output of the actor network consists of the mean and standard deviation of a Gaussian distribution. The action is sampled from this distribution. When starting with a large standard deviation, exploration of actions is encouraged. With the random samples, we do not have the guarantee that all state-action pairs are visited. However, the benefit of using neural networks as approximators is that they can generalize the learnings from similar states and actions.

In every training iteration, a buffer is filled with states, actions, and (scaled) rewards following the current (initially random) policy $\pi(s_t; \theta)$. With these observations, the advantage is calculated, which is an essential indication of whether the action that was taken is better or worse than the average policy action (i.e., the output of the critic network). This advantage calculation is done using the Generalized Advantage Estimation method as described by Schulman et al. (2015). This gives an indication of the direction in which the weights of the actor network need to be updated.

Based on the advantage information, the parameters of both neural networks are updated by computing a loss function, based on a trajectory of following the current policy. The updates aim to improve the policy and the value estimates. This means that for the actor network, we need to find the direction of choosing better actions. The loss function of Schulman et al. (2017) is used for the actor network, and it differs from other DRL algorithms in the sense that it clips the loss, meaning that the updates of the policies are not taking excessive steps. For the critic network, we need to close the gap between the observed and predicted values (mean squared error loss). Updating the neural network weights by filling the buffer and computing the loss functions constitutes one iteration. Typically the learning continues until some convergence is reached, meaning that we no longer observe significant improvements in the rewards achieved by consecutive policies.

4.2 Algorithm parameters

For many steps in the PPO algorithm, the values for the hyperparameters need to be selected. First of all, the architecture of the neural networks needs to be defined, which consists of the number of hidden layers, the number of neurons in these hidden layers, and the activation function. The number of neurons of the input layer is given by the size of the state space [i.e., the number of different combinations of variable values in (1)], whereas the number of neurons of the output layer is equal to the size of the action space [i.e., the number of different combinations of variable values in (2)], multiplied by two as we want to output the mean and standard deviation of the Gaussian distribution for each action. Additionally, we need to determine the method for initializing the weights of the network parameters.

After initializing the neural networks, decisions about the length of the buffer and the parameters (γ, λ) used for calculating advantages are chosen. The γ parameter corresponds to the discount rate mentioned in Sect. 3, while the λ controls the trade-off between bias and variance in the estimation of the advantage function. For updating the networks, we need to identify the relevant parameter for the loss function, namely the ϵ that indicates how far away from the 'old' policy the network is allowed to be after an update. To update the network, we use the popular Adam optimization procedure to update the network weights, instead of classical stochastic gradient descent. The key hyperparameter for this procedure is the learning rate. Finally, the networks are updated using minibatches of a certain size and several update epochs that are performed with the same training data.

We implemented the 'standard' PPO algorithm as described by Schulman et al. (2017), using the same hyperparameters presented in their paper, for which they concluded the PPO algorithm to perform well in a number of different problem environments. Although hyper-parameter tuning might improve the results, we decided to use these given parameter settings for each of the three inventory cases, to investigate how this PPO algorithm works without the time-consuming hyper-parameter tuning steps. However, to make the algorithm applicable to the three different inventory cases under consideration, we had to choose how to represent and scale the states, actions, and rewards. We elaborate on this in the following subsections.

A summary of the used hyperparameters for the PPO algorithm is given in Table 4. Experimentation on these parameters confirmed the suitability of the parameters found by Schulman et al. (2017).

4.3 State space transformation

As we use neural networks as approximators for the policy and value function, we perform a common preprocessing step to normalize the input for the neural network. This normalization prevents vanishing or exploding gradients, improves convergence and generalization, and helps with regularization. The values of the state variables (1) are normalized to a range of $[-1, 1]$, to keep the values in a relevant domain for the hyperbolic tangent activation function of the neural network. For the general case, the values are normalized to a range of $[0, 1]$ as opposed to $[-1, 1]$. This enables us to get an output of the untrained network that lies in the interval of $[-1, 1]$, such that the actions do not have to be clipped in the beginning.

A neural network is initialized with small values and updated with small gradient steps. If the input values are not normalized, it takes a long time for the weights of the actor network to converge to well-performing action values and for the weights of the critic network to converge to accurate value functions. We want the interval of the state variables before normalizing to have a wide range, to ensure we do not limit our method in its observations. The lower bound is 0 for every variable

Table 4 PPO hyper-parameters

Hyper-parameter	Parameter value
# of iterations	10,000 (Linear, Divergent); 50,000 (general)
Layers	2 hidden layers with 64 nodes each: (64, 64)
Activation function	tanh
Initialization of weights	Glorot-Uniform
Buffer size	256
Discount factor (γ)	0.99
GAE parameters (λ)	0.95
ϵ	0.2
Adam learning rate	1e−4
Batch size	64
Update epochs	10

because we do not have orders, backorders, or shipments that can be negative. With a large scaling upper bound, the state space values will be very close together, which also makes it hard for the neural network to distinguish different states from each other. Therefore, we test the performance of several different scaling upper bounds and assess which one performs the best in learning well-performing values.

For the simple network structures (i.e., the linear and divergent cases), we expect that transforming the input variables for the neural network, without limiting the transition dynamics with the upper bounds, will work reasonably well to find well-performing policies. However, as the problem grows in complexity (i.e., increasing state and action space), exploration may get trapped in unlikely states, from which no recovery is possible. Consider for example a situation where there is an extreme amount of inventory. If the algorithm takes actions to get out of this situation, by not ordering replenishments, the positive reinforcement signals could have a long delay, because of the slow depletion of inventory. One option to avoid this is to use the scale on the state variables both for normalizing the input to the neural network, as well as for limiting the variables in the transition function. We evaluate this 'transition-limit' option for the general case, as it is the most complex case and could benefit from this adjustment. It is important to note that the upper bounds should only be used in training the algorithm, not in evaluating the performance of the trained policy.

4.4 Action space transformation

For all inventory cases, the action to take is the order quantity at the upstream location. Reinforcement learning problems can either have a discrete or continuous action space. For every output node, the neural network of the actor outputs either the probability that this action will be executed (discrete actions) or the action-value (continuous actions). With a discrete action space, the agent decides which distinct action to perform from a finite action set. For example, when we have four edges (linear case), and we want the order size to be in the interval of [0, 30], we have $31^4 = 923,521$ different actions. For the case of the CardBoard Company using the same interval, this would even result in an astonishing number of $31^{18} = 6.99 \cdot 10^{26}$ actions. With this number of actions, the neural network cannot even be initiated on most computers due to memory limits, as for every distinct action in this action set, the neural network has a separate output node. The probability of taking one of the actions is computed by a *softmax* function on the output layer. Because this results in a large network that is not scalable to larger action spaces, we choose to implement a continuous action space.

With a continuous action space, the number of output nodes of the neural network is equal to twice the number of different combinations of variable values in the action space. In (2), we defined an order quantity per edge of the network. The output of the neural network is the mean and standard deviation of the order quantity for every edge (5). The action for each edge is then sampled from a Gaussian distribution using these parameters. This action space results in a more scalable neural network, as the number of output nodes only grows linearly with the number of edges in the network, rather than exponentially in the case of a discrete action space.

$$a_t = \left[\mu_{w,t}, \sigma_{w,t}, \mu_{r,w,t}, \sigma_{r,w,t} \right] \qquad \forall w \in W, r \in R \tag{5}$$

In the linear and divergent cases, the number of edges is equal to the number of stock points. However, in the general case, the number of edges (18) is much larger than the number of stock points (9). This may result in issues with scalability. To explore the impact of reducing the size of the action space, we consider two different action space implementations for the general case: (1) order per edge and (2) order per stock point. In the case where we define an order per stock point, we use a probability to determine the predecessor stock point to which the order is sent. For example, with 60% probability, the order of retailer r_1 is sent to warehouse w_1, and with 40% probability the order is sent to warehouse w_2, meaning the order quantity is not split over multiple suppliers. These percentages are determined based on historical data of the CardBoard Company. While this action space implementation might work well in improving the scalability of the algorithm, the performance could be jeopardized, as we add an additional source of uncertainty to the problem (i.e., the random choice of supplier), which is a common challenge for DRL algorithms (Dulac-Arnold et al. 2021).

The default output of an untrained network is 0, with a standard deviation of 1. It could, therefore, take a long time to reach the desired mean output. For example, if the optimal order size is 20, the neural network has to adjust its output to 20, but might do this in very small steps. Hence, it is recommended to scale the action space to the interval $[-1, 1]$ (Raffin et al. 2019). This should speed up the learning of the network. The interval of the action space can be determined based on experimentation or reasoning on what would be the maximum allowed action in the environment. Because the action space output of the neural network is unbounded, we clip the outputs that lie outside the given interval. This way, we encourage the network to stay within the limits of the interval. We have also experimented with giving the network a penalty whenever the outputs were outside the given interval, but this resulted in worse performance.

4.5 Reward function transformation

PPO tries to maximize the rewards. Because we want to minimize our costs, we multiply the costs (3) by -1 to represent the rewards. As mentioned in Sect. 4, the PPO algorithm uses an Actor-Critic architecture. While the actor outputs the values of the actions, the critic estimates the value function. The network weights are then optimized to achieve minimal loss (MSE) between the network outputs and training data outputs. If the scale of our network outputs is significantly different in weight and bias from that of our input features, the neural network will be forced to learn unbalanced distributions of weight and bias values, which can result in a network that is not able to learn properly. To combat this, it is recommended to scale the output values (Muccino 2019). In our experiments, we noticed that the scaling of these rewards is not trivial and no best practices are defined yet. In our case, dividing the results by 1000 yielded good results and we saw that the expected value of the critic corresponded to the average reward that was gained in the experiments.

5 Experiments and results

Experiments were executed to assess the performance of the PPO algorithm from Sect. 4 in the multi-echelon inventory problem as described in Sect. 3. In this section, we first introduce the settings of the three inventory cases we experiment with (Sect. 5.1). Thereafter, we report the performance of the PPO algorithm in terms of minimizing the costs compared to the benchmarks in the linear (Sect. 5.2), divergent (Sect. 5.3), and general inventory network (Sect. 5.4). We present the cost and computational performance of the algorithm, and we provide an analysis of how to interpret the resulting policy in practice.

5.1 Experimental settings

The linear supply chain case is based on the synthetic problem settings as described by Chaharsooghi et al. (2008). The lead time is drawn from a random distribution each period and is identical for all stock points. Furthermore, the demand at the retailer is drawn from a discrete uniform distribution, and there are four stock points. We have replicated their simulation and applied our reinforcement learning method to this simulation. As a benchmark, we used their original method, the Reinforcement Learning Ordering Mechanism (RLOM).

The divergent supply chain case is based on the synthetic problem settings as described by Kunnumkal and Topaloglu (2011). The lead time here is deterministic, while the demand at the retailer follows a Poisson distribution. The mean of the Poisson distribution is drawn from a Uniform distribution. There are again four stock points, and in this case, unlike the others, there are no backorders for the warehouses. To benchmark this system, we have used the decomposition-aggregation heuristic of Rong et al. (2017). This heuristic is also considered as benchmark by Harsha et al. (2021).

The general supply chain case is inspired by a real-world case at the Card-Board Company, with nine stock points. It is structured as shown in Fig. 1, where the four warehouses in this case are paper mills that process the raw materials into paper and cardboard products. Then this intermediate product is shipped to corrugated plants (which can be considered as the five retailers), which process the intermediate product into the final products that are sold to customers. As the general network structure is already complex to study, we assume deterministic lead times in this case. In this study, we assume the demand is drawn from a Poisson distribution. The warehouses do observe backorders, however, there are no backorder costs associated with this, since only the demand from the end customers is 'penalized' for not being readily available. In the current inventory policy (the benchmark), the aim is to reach a 98% fill rate at all stock points, meaning that also here the aim is to minimize backorders between retailers and warehouses. However, the general intuition at the CardBoard Company is that improvements are possible in terms of decreasing the total amount of inventory while keeping customers satisfied.

All case specific parameters, which originate from the cases studied by Chaharsooghi et al. (2008) and Kunnumkal and Topaloglu (2011), can be found in Table 5. Please note that these benchmarks are all non-iterative, and therefore each provides only one output value. Hence, the benchmarks are depicted as straight lines in the figures with results. Unfortunately it was not possible to use one of the other value-based DRL methods we found in literature as a benchmark. As noted by Lynnerup et al. (2019), DRL studies are notoriously difficult to replicate due to the algorithms' intrinsic variance, the stochasticity in the environment, and the various hyper-parameters that can be chosen (and that are not always reported). Since the problems considered in other studies vary from our problem settings, the corresponding DRL methods cannot easily be applied to our case. Instead, we focus on explaining the approach and implementation details of our PPO algorithm, and demonstrating its applicability to different problems.

In the linear and divergent case, we run the PPO algorithm for 10,000 learning iterations. For the general case, we use 50,000 iterations as the state and action space are larger. To study the convergence of the algorithm in terms of the performance of the learned policy, we simulate the learned policy at every 100 learning iterations (at every 500 iterations for the general case). At each of these evaluation moments, 100 simulation runs are performed. In the linear case, 35 time periods are simulated to compare to the benchmark. Note that the linear case benchmark focuses on a finite horizon setting, whereas we aim to minimize the long-term average costs in the other network structures. In the divergent case, 75 time periods are simulated, and the first 25 are discarded as warm-up periods. For the general case, 100 time periods are simulated, and the first 50 are discarded as warm-up. The differences in time periods are a result of the different benchmarks that are considered. Since there is randomness in running the PPO algorithm, ten different random seeds (referred to as 'replications') are used in each case to demonstrate the consistency in performance of the algorithm. The average performance of these random seed replications is reported. The experiments were conducted on an Intel(R) Xeon(R) Processor CPU @ 2.20 GHz with 4 GB of RAM.

As indicated in Sect. 4, the state and action space need to be scaled to an appropriate range for the specific problem environment. Since our problem environments differ significantly in terms of the number of nodes and thus in the reasonable maximum amounts of total inventory and replenishment quantity, we use different scales for the state and action variables in each case. The upper bounds for the state and action space were found after experimenting with several values, and the resulting bounds are shown in Table 6.

Table 5 Parameters per case

Structure	λ_w	$\lambda_{w,r}$	d_i	K	h_w	h_r	b_w	b_r
Linear	Unif{0, 4}	Unif{0, 4}	Unif{0, 15}	4	1	1	2	2
Divergent	1	1	Pois(Unif{5, 15})	4	1	1	0	19
General	1	1	Pois(15)	9	0.6	1	0	19

Table 6 Upper bound of the state and action variables per case

Structure	ti	tb	$I^{end}_{w,t}$	$I^{end}_{r,t}$	$BO_{r,w,t}$	$BO_{r,t}$	$IT_{w,t,t'}$	$IT_{w,r,t,t'}$	$O_{w,t}$	$O_{r,w,t}$
Linear	4000	4000	1000	1000	1000	1000	150	150	30	30
Divergent	1000	450	250	250	150	150	300	75	300	75
General	4500	8250	500	500	500	250	150	75	150	75

5.2 Linear network

We first show the results of the linear network, providing insight into the applicability of the PPO algorithm to the simplest network structure. One replication of the PPO algorithm takes on average 1 h to complete. Note that the computation time of the algorithm can be reduced with early stopping procedures once the solution has converged.

The results of the experiments can be found in Fig. 2. As a benchmark, we have included the average results of the RLOM approach of Chaharsooghi et al. (2008).

We see that the PPO algorithm improves quickly and it only needs a few iterations to improve on the benchmark (note that the benchmark also relies on RL). After completing training, the PPO algorithm results in 16.4% lower costs compared to the benchmark. From this, we conclude that the PPO algorithm succeeded in yielding good rewards while using a large state- and action space, and is able to learn well-performing actions. These results indicate that deep reinforcement learning might be a promising method for other more complicated cases.
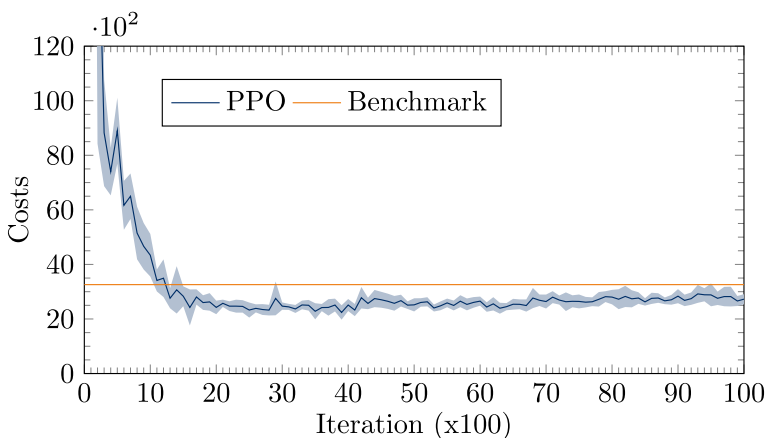
Fig. 2 Performance of the PPO algorithm in the linear inventory system. The blue shade depicts the 95% confidence interval (colour figure online)

5.3 Divergent network

In this section, we present the results of applying the PPO algorithm to the divergent network case. One full replication of the deep reinforcement learning method takes on average 1 h. The result of the deep reinforcement learning method can be seen in Fig. 3.

To benchmark our method, we use the decomposition-aggregation (DA) heuristic of Rong et al. (2017), which is proven to be asymptotic optimal. Besides the DA heuristic, Rong et al. (2017) also propose the Recursive-Optimization heuristic; they conclude that both heuristics perform well, but that DA is significantly less computationally intensive. With this heuristic, we find the base-stock levels [124, 30, 30, 30]. Hence, the base-stock level for the warehouse should be 124, while the base-stock level for the retailers is 30. The calculations for the heuristic can be found in Appendix A. The average costs of the benchmark are 4059.

As can be seen in Fig. 3, PPO is able to learn the required actions quickly, as the costs decrease vastly over the first iterations. After only 3000 iterations, which takes about 30 min, the method is able to beat the benchmark. At iteration 6300, the algorithm has learned the lowest costs, with an average cost performance of 3600, which means that the method is able to beat the benchmark by an impressive 11.3%. Note that it is not recommended to continue learning after convergence has been reached, as the performance could deteriorate after more learning iterations due to overfitting. From these results, we conclude that the deep reinforcement learning method can easily be applied to other cases, without any alterations to the algorithm. There might be still some improvement possible by tuning the parameters of the algorithm. However, this is a non-trivial and time-consuming task (Gijsbrechts et al. 2022). Therefore, we decided to keep the current parameter values. With the current settings, the method is already able to beat a base-stock level heuristic.

To gain more insights into the decisions that the deep reinforcement learning method makes, and to interpret the resulting policy when being used in practice,
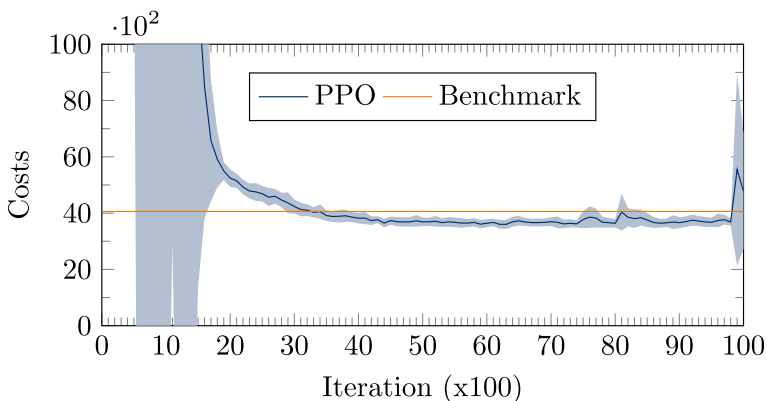

Fig. 3 Performance of the PPO algorithm in the divergent inventory system. The blue shade depicts the 95% confidence interval (colour figure online)

we look at how the actions of the method depend on the state. To do this, we use the trained network and feed the different states as input to this network. The output of the network is the action that it will take, i.e., the quantity of the order it will place upstream. When we visualize these states and actions in a heatmap, we see how the actions are coherent with the states. Because we use a two-dimensional heatmap, we can only vary two different variables in the state vector at a time, so the other values in the state vector remain fixed. This value is based on the average value of the particular state variable, which we determined using simulation.

The heatmap of Fig. 4a shows the cohesion of the inventory in the warehouse with the number of items in transit for the warehouse. We see that the order quantity gets lower as the inventory of the warehouse gets higher. The order quantity only slightly depends on the number of items in transit. When the number of items in transit is low, the order quantity is high. This order quantity slightly decreases when the number of items in transit increases. However, when the number of items in transit passes 50, the order quantity tends to increase again slightly. Although this is counter intuitive, it might be the result of providing only a limited number of observations of these states to the neural network, as they do not occur often in the simulation, so they seem to be unlikely to occur in practice. All in all, the heatmap shows that PPO is able to make explainable connections between states and actions, but also resulted in some actions that make less sense.

The state-action connections of the retailers can be found in Fig. 4b. For these heatmaps, we investigated the different retailers and the relation between their inventory and both the inventory of the warehouse and the number of items in transit. We see that there is no clear dependency between the actions and the inventory of the warehouse. This makes sense, as the number of items in the warehouse does not directly affect the inventory position of the retailer. The connection between the number of items in transit and the inventory of the retailer is clearly visible. The resulting policy does not order when the number of items in transit is large. Also, the current inventory of the retailers impacts the order quantity. Especially for retailers 1 and 2, the order quantity quickly increases when the retailers have backorders (with a slower increase for retailer 3).
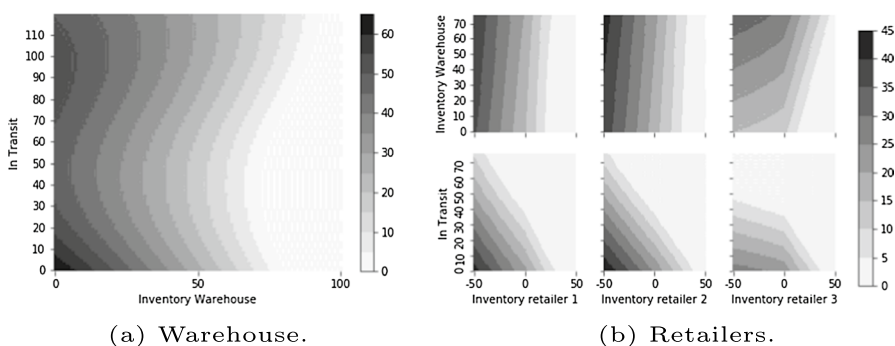

(a) Warehouse.   (b) Retailers.

Fig. 4 Actions of the warehouse and retailers

With this analysis, we demonstrate that the PPO algorithm finds sensible policies, and we provide guidance on how to interpret the resulting policy in practice. Although it is not possible to quickly see the relations between all the different states in these two-dimensional heatmaps, we do get insights into some important connections.

5.4 General network

In this section, we present the results of applying the PPO algorithm to the general network case. We conducted three different sets of experiments. In Experiment set 1, we defined the action space with orders per stock point. Experiment set 2 had a larger action space, as an order per edge was considered (see Sect. 4.4). To ensure that the PPO algorithm does not become trapped in unlikely state spaces, we defined Experiment set 3, where the transition function was restricted (see Sect. 4.3) and where we again use the action space with orders per edge.

As a benchmark, we used the actual policy of the CardBoard Company, which aimed to achieve a fill rate of 98% for their customers. They aimed to achieve this by also enforcing the fill rate of 98% for the connections between the paper mills and corrugated plants, which is similar to a decentralized inventory policy. We set up a benchmark by manually tuning the base-stock parameters such that the fill rate of 98% is met for every connection. We measure this fill rate by running a simulation model of the network for 100 time periods, where we remove the first 50 periods to account for the warm-up period and replicated this simulation 500 times. Using this method, we obtained the following base stock parameters for Experiment 1: [82, 100, 64, 83, 35, 35, 35, 35, 35], resulting in an average total cost of 10,467. For Experiments 2 and 3, the base stock parameters were defined as: [37, 47, 33, 63, 30, 30, 30, 30, 30], with an average total cost of 4,797. There is a difference between these base stock parameters due to the higher uncertainty in Experiment set 1 as a result of the randomness in where the upstream order will be placed. In Experiment set 2 and 3, the orders are split across all the upstream connections, resulting in lower demand uncertainty for the warehouse stock points, higher delivery reliability between warehouse and retail stock points, and thus lower base-stock parameters.

We run the deep reinforcement learning method for 50,000 iterations for the three experiment sets, which takes approximately 5.8 h per replication. Table 7 shows the most important results of every experiment set. When looking at the results, we observe that the PPO algorithm is not able to achieve consistent results for both

Table 7 Results of the benchmark and the PPO algorithm in the general inventory system

Experiment	Benchmark	Base-stock parameters	Best performance	Average performance
Set 1	10,467	[82, 100, 64, 83, 35, 35, 35, 35, 35]	8714	630,401
Set 2	4797	[37, 47, 33, 63, 30, 30, 30, 30, 30]	4175	314,923
Set 3	4797	[37, 47, 33, 63, 30, 30, 30, 30, 30]	3935	4481

Experiment set 1 and 2. Although the lowest costs are lower than the benchmark, the average costs are high, due to some runs that are not able to converge. Experiment set 3, which implements the transition limit, is able to converge in every run and is therefore yielding an average performance that is lower than the benchmark. For the remainder of this section, we will further analyze the results of Experiment 3.

Figure 5 shows the result of Experiment 3, compared with the benchmark. We see that the PPO algorithm once again improves quickly in this case, but needs some time to converge and yield steady results. After 22,000 iterations, which takes about 2.5 h, the average costs drop below the benchmark for the first time. On average, the PPO algorithm improves the benchmark by 6.6%. To gain more insight into the decision that the PPO algorithm makes, we will look into the fill rate of the inventory system and compare it with the benchmark. Note that for the case of the CardBoard Company, we are not able to create heatmaps, as we have too many state variables that depend on each other.

For every replication of our deep reinforcement learning method, we use the fully trained network to run a simulation for 100 periods. We then remove the first 50 periods, to account for the warm-up period. Figure 6 shows the average fill rate of replication 1 for both the benchmark and our deep reinforcement learning method. While we did not impose the 98% fill rate as a constraint in our model, the method is still meeting this requirement for almost every retailer. We see that the method did learn that it is not necessary to achieve a 98% fill rate for every stock point, but only for the retailers, as they are directly delivering to the customers. With respect to the warehouses, our PPO algorithm decides to only store inventory at warehouse 4. According to Fig. 1, this warehouse has a connection to every retailer. The PPO algorithm learned that storing inventory at only one warehouse minimizes the variability and achieves higher downstream fill rates. With this approach, the PPO algorithm has learned to simplify the general network structure of our case to a divergent network structure. While this is a valid solution to our provided case, this might not be a realistic solution in a real setting, for example, due to the distances between
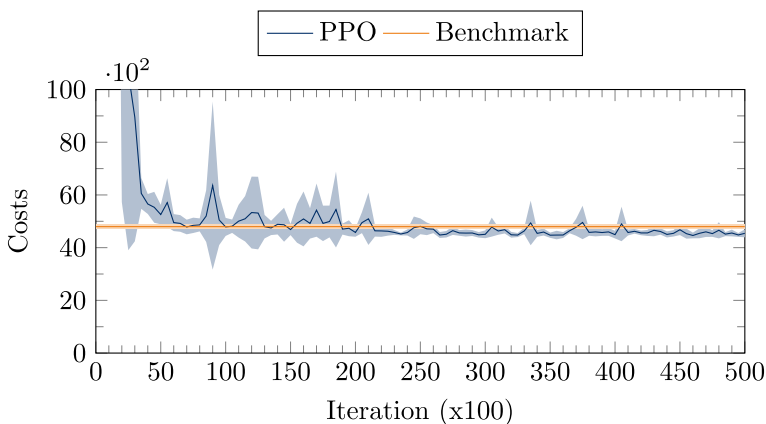

Fig. 5 Results of the PPO algorithm on the case of CardBoard Company. The blue shade depicts the 95% confidence interval (colour figure online)
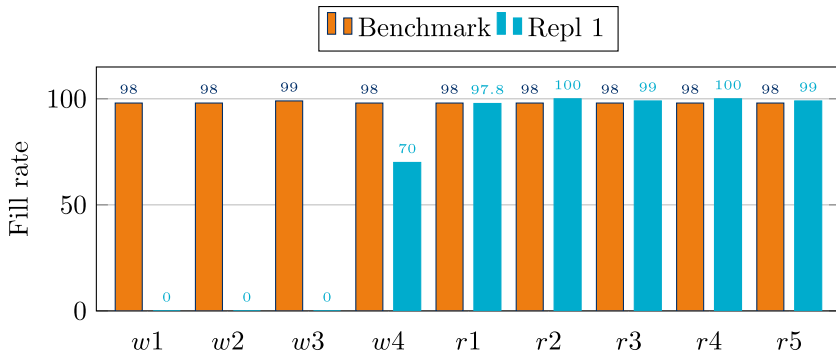
Fig. 6 Average fill rates (in percentage) for the stock points of the CardBoard Company

warehouses and retailers. Therefore, future research could focus on altering the case, such that it becomes less beneficial to only store inventory at warehouse 4. This could be done by either adding an extra long lead time to this warehouse, making using of preferred suppliers, or changing replenishment costs for different suppliers.

Based on these experiments, we conclude that the PPO algorithm is not able to converge in complex environments without using a transition limit. The extra stochasticity in Experiment set 1 results in a highly unstable method. Experiment set 2 shows that, while the stochasticity is reduced, the method remains unstable. The transition limit in Experiment set 3 ensures that the PPO algorithm does not get stuck in unlikely states and pushes the method in the right direction.

6 Conclusion

We investigated the applicability of Deep Reinforcement Learning (DRL) for finding replenishment order quantities in multi-echelon inventory management. More specifically, we proposed a PPO algorithm, a method that performs well with robust hyper-parameter values found in the literature. We applied our method to three different network structures, to show its general applicability. When applying the PPO algorithm to the different cases, the main modifications we made were related to the definition of the state and action spaces, and setting the bounds. The bounds on state and action variables have a big impact on the performance, but with minor experimentation, suitable bounds can be identified. Future research could look into automating and optimizing the process for determining these bounds. The only other modification concerns the transition limits in the larger problems to ensure a stable performance of the algorithm.

The PPO algorithm outperforms the benchmarks in all cases. In the linear case, the algorithm achieved 16.4% lower costs compared to the benchmark of Chaharsooghi et al. (2008). In the divergent network structure, PPO resulted in 11.3% lower costs compared to the benchmark of Rong et al. (2017). In the general network structure, we considered a real-world case of Cardboard Company, where we

found that a transition limit was necessary to let the PPO algorithm converge. In this general network structure case, the PPO algorithm was able to reduce the costs by 6.6%. The PPO algorithm learned to minimize variability and improve downstream fill rates by storing inventory at only one warehouse. However, this may not be a feasible solution in real-world settings. Future research could explore ways to make it less beneficial to store inventory at one warehouse by introducing more complexity and uncertainty, such as preferred suppliers or different lead times per connection. Furthermore, the PPO algorithm is executed within reasonable computation times of around 1 h, with larger problems taking more time to train. Once trained, the neural networks can instantly provide the action to take. We showed that insights into the resulting policy of the algorithm can be retrieved by creating heatmaps and investigating the prescribed actions in different states.

Our contributions to this research are threefold. First, we applied a DRL method with a continuous action space, which is not yet widely used, to a multi-echelon inventory system, creating better scalability to larger problems. If discrete actions would be used, we would not have been able to initiate the neural network for the case of the CardBoard Company. Second, we investigated a general inventory system, which has received less attention in the literature due to its complexity and intractability. Nevertheless, the relevance of a general network structure is underlined by our real-world case of the CardBoard Company, and we showed that the PPO algorithm can improve inventory policies. Third, we demonstrated the flexibility of the PPO algorithm by applying it to a variety of cases without major modifications and without any parameter tuning.

While demonstrating the applicability of the PPO algorithm in a general network structure, we investigated the performance in a simulated environment, using stationary demand distributions. Investigating the performance and suitability in a real-life environment is essential before implementing the PPO algorithm in practice. While our case reflects a real-life environment, further research could focus on implementing more real-life aspects. Additionally, more research should be done on the explainability of our method and deep reinforcement learning in general. As the explainability aspect is a key component of implementing such methods in practice, it deserves more attention. van Hezewijk et al. (2022) provide an example of how the neural network 'black box' can be opened to interpret the resulting policy of the PPO algorithm. They do so by using multiple linear regression to find relationships between the state and action variables of the trained actor, thereby providing insight into the logic behind selecting certain actions when observing certain states.

Appendix A: Calculation of the base-stock levels for the divergent case

In this appendix, we elaborate on the calculations for the heuristic of Rong et al. (2017) to determine the base-stock levels of the divergent inventory system. The formulas and explanatory notes of Rong et al. (2017) are extended with our own calculations.

Under the decomposition-aggregation (DA) heuristic, we decompose the divergent inventory system into three two-location serial systems, solve the base-stock levels of the locations in each serial system, and aggregate the solutions utilizing a procedure that Rong et al. (2017) have called"backorder matching". We use s_i^a to denote the local base-stock level at location i based on the DA heuristic. Next, we provide a detailed description of the steps of the heuristic.

Step 1 Decompose the system into N serial systems. Serial system i consists of the warehouse and retailer i. We use 0_i to refer to the warehouse in serial system i. Utilizing the procedure from Shang and Song (2003), we approximate the echelon base-stock levels of retailer i, S_i^{SS}, and the warehouse in serial system i, $S_{0}^{SS}i$, as follows:

$$S_i^{SS} = F_{D_i}^{-1}\left(\frac{b_i + h_0}{b_i + h_i}\right) = F_{D_i}^{-1}(0.98) = 30$$
$$S_{0_i}^{SS} = \frac{1}{2}\left[G_{\tilde{D}_i}^{-1}\left(\frac{b_i}{b_i + h_i}\right) + G_{\tilde{D}_i}^{-1}\left(\frac{b_i}{b_i + h_0}\right)\right]\frac{1}{2}\left[G_{\tilde{D}_i}^{-1}(0.95) + G_{\tilde{D}_i}^{-1}(0.97)\right] = 52$$

(A1)

Here, $\tilde{D}_i$ is the total leadtime demand in serial system i. D_i is a Poisson random variable with rate $\lambda_i(L_0 + L_i)$. F^{-1} is the inverse Poisson cumulative distribution function (CDF). For the warehouse, instead of using F^{-1}, we opt instead to use G_D^{-1}, the inverse function of an approximate Poisson CDF. Let $\lceil x \rceil$ and $\lfloor x \rfloor$ be the smallest integer no less than x and the largest integer no greater than x, respectively. We define this approximate CDF of a Poisson random variable D to be the following continuous, piecewise linear function:

$$G_D(x) = \begin{cases} 2F_D(0)x, & \text{if } x \leq 0.5 \\ \begin{aligned} &F_D(\lfloor x - 0.5\rfloor) + \left[F_D(\lceil x - 0.5\rceil) \\ &- F_D(\lfloor x - 0.5\rfloor)\right](x - 0.5 \\ &- \lfloor x - 0.5\rfloor) \end{aligned} & \text{if } x \geq 0.5 \end{cases}$$

(A2)

Step 2 Calculate the local base-stock level for the warehouse in serial system i by $s_{0_1}^d = S_{0_1}^{SS} - S_i^{SS}$. For the retailers, we have $s_i^d = S_i^{\zeta S}, \forall i \in \{1, 2, \dots, N\}$. We approximate the expected backorders of the warehouse in serial system i by:

$$E[B_{0_i}] = E\left[\left(D_{0_i} - s_{0_i}^d\right)^+\right] = Q_{D_{0_i}}\left(s_{0_i}^d\right) = 0.97949$$

(A3)

In (A3), D_0 is a Poisson random variable with rate $\lambda_i L_0$, and $Q_D(x)$ is the loss function of the Poisson random variable D.

Step 3 Aggregate the serial systems back into the OWMR system utilizing a "backorder matching" procedure. We approximate the total expected backorders at the warehouse by $E[B_0] \cong \Sigma_{i=1}^N E[B_0]$. Specifically, the backorder matching procedure sets s_0^a, the base-stock level at the warehouse, equal to the smallest integer s_0 such that:

$$E\left[\left(D_0 - s_0\right)^+\right] \leq \sum_{i=1}^{N} E\left[B_{0_i}\right] = 2.9384 \qquad (A4)$$

$$s_0^a = Q_{D_0}^{-1}\left(\sum_{i=i}^{N} Q_{D_{0_i}}\left(s_{0_i}^d\right)\right) = 124 \qquad (A5)$$

$Q_D^{-1}(y)$ is defined as $\min\left\{s \in \mathbb{Z} : Q_D(s) \leq y\right\}$.

This gives us the base-stock parameters: [124, 30, 30, 30].

Declarations

References

Arnold J, Chapman S, Clive L (2008) Introduction to materials management. Pearson Prentice Hall, Hoboken

Arulkumaran K, Deisenroth MP, Brundage M, Bharath AA (2017) Deep reinforcement learning: a brief survey. IEEE Signal Process Mag 34(6):26–38. https://doi.org/10.1109/MSP.2017.2743240

Bellman R (1957) Dynamic programming. Princeton University Press, Princeton

Ben-Tal A, Golany B, Shtern S (2009) Robust multi-echelon multi-period inventory control. Eur J Oper Res 199(3):922–935. https://doi.org/10.1016/j.ejor.2009.01.058

Bertsimas D, Thiele A (2006) A robust optimization approach to inventory theory. Oper Res 54(1):150–168. https://doi.org/10.1287/opre.1050.0238

Carlos DPA, Jairo RMT, Aldo FA (2008) Simulation-optimization using a reinforcement learning approach. In: Winter simulation conference, pp 1376–1383. https://doi.org/10.1109/WSC.2008.4736213

Chaharsooghi SK, Heydari J, Zegordi SH (2008) A reinforcement learning model for supply chain ordering management: an application to the beer game. Decis Support Syst 45(4):949–959. https://doi.org/10.1016/j.dss.2008.03.007

Chen F, Song J-S (2001) Optimal policies for Multiechelon inventory problems with Markov-modulated demand. Oper Res 49(2):226–234. https://doi.org/10.1287/opre.49.2.226.13528

Chen F, Feng Y, Simchi-Levi D (2002) Uniform distribution of inventory positions in two-echelon periodic review systems with batch-ordering policies and interdependent demands. Eur J Oper Res 140(3):648–654. https://doi.org/10.1016/S0377-2217(01)00203-X

Çimen M, Kirkbride C (2013) Approximate dynamic programming algorithms for multidimensional inventory optimization problems. IFAC Proc Vol IFAC-PapersOnline 46(9):2015–2020. https://doi.org/10.3182/20130619-3-RU-3018.00441

Çimen M, Kirkbride C (2017) Approximate dynamic programming algorithms for multidimensional flexible production-inventory problems. Int J Prod Res 55(7):2034–2050. https://doi.org/10.1080/00207543.2016.1264643

De Kok AG, Grob C, Laumanns M, Minner S, Rambau J, Schade K (2018) A typology and literature review on stochastic multi-echelon inventory models. Eur J Oper Res 269(3):955–983. https://doi.org/10.1016/j.ejor.2018.02.047

Dittrich M-A, Fohlmeister S (2020) A deep q-learning-based optimization of the inventory control in a linear process chain. Prod Eng 15:1–9. https://doi.org/10.1007/s11740-020-01000-8

Dogru MF, de Kok AG, van Houtum GJ (2013) Newsvendor characterizations for one-warehouse multi-retailer inventory systems with discrete demand under the balance assumption. Cent Eur J Oper Res 21:541–559. https://doi.org/10.1007/s10100-012-0246-7

Dulac-Arnold G, Levine N, Mankowitz DJ, Li J, Paduraru C, Gowal S, Hester T (2021) Challenges of real-world reinforcement learning: definitions, benchmarks and analysis, vol 110. Springer, US. https://doi.org/10.1007/s10994-021-05961-4

Elson Kosasih E, Brintrup A (2021) Reinforcement learning provides a flexible approach for realistic supply chain safety stock optimisation. arXiv e-prints arXiv:2107.00913 [cs.MA]

Ganeshan R (1999) Managing supply chain inventories: a multiple retailer, one warehouse, multiple supplier model. Int J Prod Econ 59(1):341–354. https://doi.org/10.1016/S0925-5273(98)00115-7

Geng W, Qiu M, Zhao X (2010) An inventory system with single distributor and multiple retailers: operating scenarios and performance comparison. Int J Prod Econ 128(1):434–444. https://doi.org/10.1016/j.ijpe.2010.08.002

Giannoccaro I, Pontrandolfo P (2002) Inventory management in supply chains: a reinforcement learning approach. Int J Prod Econ 78(2):153–161. https://doi.org/10.1016/S0925-5273(00)00156-0

Gijsbrechts J, Boute RN, Van Mieghem JA, Zhang D (2022) Can deep reinforcement learning improve inventory management? Performance and implementation of dual sourcing-mode problems. Manuf Serv Oper Manag. https://doi.org/10.1287/msom.2021.1064

Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT Press, Cambridge

Gumus AT, Guneri AF (2007) Multi-echelon inventory management in supply chains with uncertain demand and lead times: literature review from an operational research perspective. Proc Inst Mech Eng Part B J Eng Manuf. https://doi.org/10.1243/09544054JEM889

Gumus AT, Guneri AF (2009) A multi-echelon inventory management framework for stochastic and fuzzy supply chains. Expert Syst Appl 36(3):5565–5575. https://doi.org/10.1016/j.eswa.2008.06.082

Gumus AT, Guneri AF, Ulengin F (2010) A new methodology for multi-echelon inventory management in stochastic and neuro-fuzzy environments. Int J Prod Econ 128:248–260. https://doi.org/10.1016/j.ijpe.2010.06.019

Harsha P, Jagmohan A, Kalagnanam JR, Quanz B, Singhvi D (2021) Math programming based reinforcement learning for multi-echelon inventory management. CoRR. arXiv:2112.02215

Iida T (2001) The infinite horizon non-stationary stochastic multi-echelon inventory problem and near-myopic policies. Eur J Oper Res 134(3):525–539. https://doi.org/10.1016/S0377-2217(00)00275-7

Jain S, Raghavan NRS (2009) A queuing approach for inventory planning with batch ordering in multi-echelon supply chains. Cent Eur J Oper Res 17:95–110. https://doi.org/10.1007/s10100-008-0077-8

Jiang C, Sheng Z (2009) Case-based reinforcement learning for dynamic inventory control in a multi-agent supply-chain system. Expert Syst Appl 36(3 PART 2):6520–6526. https://doi.org/10.1016/j.eswa.2008.07.036

Kalchschmidt M, Zotteri G, Verganti R (2003) Inventory management in a multi-echelon spare parts supply chain. Int J Prod Econ 81(82):397–413. https://doi.org/10.1016/S0925-5273(02)00284-0

Kim CO, Jun J, Baek J-G, Smith RL, Kim YD (2005) Adaptive inventory control models for supply chain management. Int J Adv Manuf Technol 26(9–10):1184–1192. https://doi.org/10.1007/s00170-004-2069-8

Kim CO, Kwon IH, Baek J-G (2008) Asynchronous action-reward learning for nonstationary serial supply chain inventory control. Appl Intell 28(1):1–16. https://doi.org/10.1007/s10489-007-0038-2

Kim CO, Kwon IH, Kwak C (2010) Multi-agent based distributed inventory control model. Expert Syst Appl 37(7):5186–5191. https://doi.org/10.1016/j.eswa.2009.12.073

Kunnumkal S, Topaloglu H (2011) Linear programming based decomposition methods for inventory distribution systems. Eur J Oper Res 211(2):282–297. https://doi.org/10.1016/j.ejor.2010.11.026

Kwak C, Choi JS, Kim CO, Kwon IH (2009) Situation reactive approach to vendor managed inventory problem. Expert Syst Appl 36(5):9039–9045. https://doi.org/10.1016/j.eswa.2008.12.018

Kwon IH, Kim CO, Jun J, Lee JH (2008) Case-based myopic reinforcement learning for satisfying target service level in supply chain. Expert Syst Appl 35(1–2):389–397. https://doi.org/10.1016/j.eswa.2007.07.002

Lambrecht M, Muchstadt J, Luyten R (1984) Protective stocks in multi-stage production systems. Int J Prod Res 22(6):1001–1025. https://doi.org/10.1080/00207548408942517

Lambrecht M, Luyten R, Vander Eecken J (1985) Protective inventories and bottlenecks in production systems. Eur J Oper Res 22(3):319–328. https://doi.org/10.1016/0377-2217(85)90251-6

Li J, Guo P, Zuo Z (2008) Inventory control model for mobile supply chain management. In: Proceedings—The 2008 International Conference on Embedded Software and Systems Symposia, ICESS Symposia. https://doi.org/10.1109/ICESS.Symposia.2008.85

Lynnerup NA, Nolling L, Hasle R, Hallam J (2019) A survey on reproducibility by evaluating deep reinforcement learning algorithms on real-world robots. (CoRL), pp 1–24. arXiv:1909.03772

Mes MRK, Rivera AP (2017) Approximate dynamic programming by practical examples. International series in operations research and management science, vol 248. https://doi.org/10.1007/978-3-319-47766-4_3

Minner S, Diks EB, De Kok AG (2003) A two-echelon inventory system with supply lead time flexibility. IIE Trans (Inst Ind Eng) 35(2):117–129. https://doi.org/10.1080/07408170304383

Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller MA (2013) Playing Atari with deep reinforcement learning. CoRR. arXiv:1312.5602

Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Hassabis D (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529–533. https://doi.org/10.1038/nature14236

Muccino E (2019) Scaling reward values for improved deep reinforcement learning. https://medium.com/mindboard/scaling-reward-values-for-improved-deep-reinforcement-learninge9a89f89411d

Nahmias S, Smith SA (1993) Mathematical models of retailer inventory systems: a review. In: Perspectives in operations management. Springer US, Boston, pp 249–278. https://doi.org/10.1007/978-1-4615-3166-1_14

Nahmias S, Smith SA (1994) Optimizing inventory levels in a two-echelon retailer system with partial lost sales. Manag Sci 40(5):582–596. https://doi.org/10.1287/mnsc.40.5.582

Oroojlooyjadid A, Nazari M, Snyder L, Takáč M (2017) A deep Q-network for the beer game: a deep reinforcement learning algorithm to solve inventory optimization problems. Manuf Serv Oper Manag 24(1):285–304. arXiv:1708.05924

Peng Z, Zhang Y, Feng Y, Zhang T, Wu Z, Su H (2019) Deep reinforcement learning approach for capacitated supply chain optimization under demand uncertainty. In: 2019 Chinese automation congress (CAC), pp. 3512–3517. https://doi.org/10.1109/CAC48633.2019.8997498

Pirhooshyaran M, Snyder LV (2020) Simultaneous decision making for stochastic multi-echelon inventory optimization with deep neural networks as decision makers. CoRR. arXiv:2006.05608

Raffin A, Hill A, Ernestus M, Gleave A, Kanervisto A, Dormann N (2019) Stable baselines. https://github.com/DLR-RM/stable-baselines3.GitHub

Rao U, Scheller-Wolf A, Tayur S (2000) Development of a rapid-response supply chain at Caterpillar. Oper Res 48(2):189–204. https://doi.org/10.1287/opre.48.2.189.12380

Rao JJ, Ravulapati KK, Das TK (2003) A simulation-based approach to study stochastic inventory-planning games. Int J Syst Sci 34(12–13):717–730. https://doi.org/10.1080/00207720310001640755

Rau H, Wu MY, Wee HM (2003) Integrated inventory model for deteriorating items under a multi-echelon supply chain environment. Int J Prod Econ 86(2):155–168. https://doi.org/10.1016/S0925-5273(03)00048-3

Ravulapati KK, Rao J, Das TK (2004) A reinforcement learning approach to stochastic business games. IIE Trans (Inst Ind Eng) 36(4):373–385. https://doi.org/10.1080/07408170490278698

Rong Y, Atan Z, Snyder LV (2017) Heuristics for base-stock levels in multi-echelon distribution networks. Prod Oper Manag 26(9):1760–1777. https://doi.org/10.1111/poms.12717

Saitoh F, Utani A (2013) Coordinated rule acquisition of decision making on supply chain by exploitationoriented reinforcement learning—beer game as an example—. In: Lecture Notes in Computer

Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol 8131 LNCS, pp 537–544. https://doi.org/10.1007/978-3-642-40728-4_67

Schulman J, Moritz P, Levine S, Jordan M, Abbeel P (2015) High-dimensional continuous control using generalized advantage estimation. arXiv preprint arXiv:1506.02438

Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms, pp 1–12. arXiv:1707.06347

Shang KH, Song JS (2003) Newsvendor bounds and heuristic for optimal policies in serial supply chains. Manag Sci 49(5):618–638. https://doi.org/10.1287/mnsc.49.5.618.15147

Shervais S, Shannon TT (2001) Improving theoretically-optimal and quasi-optimal inventory and transportation policies using adaptive critic based approximate dynamic programming. In: Proceedings of the international joint conference on neural networks, vol 2. IEEE, pp 1008–1013. https://doi.org/10.1109/IJCNN.2001.939498

Shervais S, Shannon T, Lendaris G (2003) Intelligent supply chain management using adaptive critic learning. IEEE Trans Syst Man Cybern Part A Syst Hum 33(2):235–244. https://doi.org/10.1109/TSMCA.2003.809214

Shin J, Lee JH (2019) Multi-timescale, multi-period decision-making model development by combining reinforcement learning and mathematical programming. Comput Chem Eng 121:556–573. https://doi.org/10.1016/j.compchemeng.2018.11.020

Sui Z, Gosavi A, Lin L (2010) A reinforcement learning approach for inventory replenishment in vendor-managed inventory systems with consignment inventory. EMJ Eng Manag J 22(4):44–53. https://doi.org/10.1080/10429247.2010.11431878

Sutton RS, Barto AG (2018) Reinforcement learning—an introduction. MIT press, Cambridge

Topan E, Eruguz AS, Ma W, Van Der Heijden MC, Dekker R (2020) A review of operational spare parts service logistics in service control towers. Eur J Oper Res 282(2):401–414. https://doi.org/10.1016/j.ejor.2019.03.026

Tunc H, Kilic OA, Tarim SA, Eksioglu B (2011) The cost of using stationary inventory policies when demand is non-stationary. Omega 39(4):410–415. https://doi.org/10.1016/j.omega.2010.09.005

Unyimadu S, Anyibuofu K (2014) Inventory management practices in manufacturing firms. Ind Eng Lett 4

van Hezewijk L, Dellaert N, Van Woensel T, Gademann N (2022) Using the proximal policy optimisation algorithm for solving the stochastic capacitated lot sizing problem. Int J Prod Res 1–24

Van Roy B, Bertsekas DP, Lee Y, Tsitsiklis JN (1997) A neuro-dynamic programming approach to retailer inventory management. In: Proceedings of the 36th IEEE conference on decision and control, vol 4. IEEE, pp 4052–4057. https://doi.org/10.1109/CDC.1997.652501

Van Tongeren T, Kaymak U, Naso D, Van Asperen E (2007) Q-learning in a competitive supply chain. In: Conference proceedings—IEEE international conference on systems, man and cybernetics, pp 1211–1216. https://doi.org/10.1109/ICSMC.2007.4414132

Vanvuchelen N, Gijsbrechts J, Boute R (2020) Use of proximal policy optimization for the joint replenishment problem. Comput Ind 119:103239. https://doi.org/10.1016/j.compind.2020.103239

Vercraene S, Gayon JP (2013) Optimal control of a production-inventory system with productreturns. Int J Prod Econ 142(2):302–310. https://doi.org/10.1016/j.ijpe.2012.11.012

Woerner S, Laumanns M, Zenklusen R, Fertis A (2015) Approximate dynamic programming for stochastic linear control problems on compact state spaces. Eur J Oper Res 241(1):85–98. https://doi.org/10.1016/j.ejor.2014.08.003

Xu J, Zhang J, Liu Y (2009) An adaptive inventory control for a supply chain. In: 2009 Chinese control and decision conference, CCDC 2009, pp 5714–5719. https://doi.org/10.1109/CCDC.2009.5195218

Yang S, Zhang J (2015) Adaptive inventory control and bullwhip effect analysis for supply chains with nonstationary demand. In; Proceedings of the 2015 27th Chinese control and decision conference, CCDC 2015, pp 3903–3908. https://doi.org/10.1109/CCDC.2015.7162605

Zarandi MHF, Moosavi SV, Zarinbal M (2013) A fuzzy reinforcement learning algorithm for inventory control in supply chains. Int J Adv Manuf Technol 65(1–4):557–569. https://doi.org/10.1007/s00170-012-4195-z

Zhang K, Xu J, Zhang J (2013) A new adaptive inventory control method for supply chains with non-stationary demand. In: 2013 25th Chinese control and decision conference, CCDC 2013, pp 1034–1038. https://doi.org/10.1109/CCDC.2013.6561076