

Combining deep reinforcement learning and multi-stage stochastic programming to address the supply chain inventory management problem

Original

Combining deep reinforcement learning and multi-stage stochastic programming to address the supply chain inventory management problem / Stranieri, Francesco; Fadda, Edoardo; Stella, Fabio. - In: INTERNATIONAL JOURNAL OF PRODUCTION ECONOMICS. - ISSN 0925-5273. - ELETTRONICO. - 268:(2024). [10.1016/j.ijpe.2023.109099]

Availability:

This version is available at: 11583/2984070 since: 2023-11-24T11:03:11Z

Publisher:

Elsevier

Published

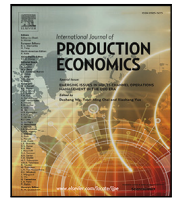
DOI:10.1016/j.ijpe.2023.109099

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)



Combining deep reinforcement learning and multi-stage stochastic programming to address the supply chain inventory management problem

Francesco Stranieri^{a,b,*}, Edoardo Fadda^c, Fabio Stella^a

^a Department of Informatics, Systems, and Communication (DISCo), University of Milano-Bicocca, Viale Sarca, 336, Milan, 20126, Italy

^b Department of Control and Computer Engineering (DAUIN), Polytechnic of Turin, Corso Duca degli Abruzzi, 24, Turin, 10129, Italy

^c Department of Mathematical Sciences (DISMA), Polytechnic of Turin, Corso Duca degli Abruzzi, 24, Turin, 10129, Italy

ARTICLE INFO

Dataset link: <https://github.com/frenkowski/SCIMAI-Gym>

Keywords:

Inventory management
Deep reinforcement learning
Stochastic programming

ABSTRACT

We introduce a novel heuristic designed to address the supply chain inventory management problem in the context of a two-echelon divergent supply chain. The proposed heuristic advances the current state-of-the-art by combining deep reinforcement learning with multi-stage stochastic programming. In particular, deep reinforcement learning is employed to determine the number of batches to produce, while multi-stage stochastic programming is applied to make shipping decisions. To support further research, we release a publicly available software environment that simulates a wide range of two-echelon divergent supply chain settings, allowing the manipulation of various parameter values, including those associated with seasonal demands. We then present a comprehensive set of numerical experiments considering constraints on production and warehouse capacities under fixed and variable logistic costs. The results demonstrate that the proposed heuristic significantly and consistently outperforms pure deep reinforcement learning algorithms in minimizing total costs. Moreover, it overcomes several inherent limitations of multi-stage stochastic programming models, thus underscoring its potential advantages in addressing complex supply chain scenarios.

1. Introduction

Supply chain inventory management (SCIM) is a critical challenge faced by several companies; it involves making decisions regarding the *number of batches* to produce at the factory and how many of them ship to each distribution warehouse. While higher production levels and inventory stocks allow companies to better meet customers' demands, they come at greater costs. Therefore, the goal of SCIM is to find a trade-off between satisfying customer demands and minimizing supply chain costs while maintaining market competitiveness (Brandimarte and Zotteri, 2007).

Although the SCIM problem is typically formulated as a multi-stage (MS) stochastic, mixed-integer linear problem (MILP), existing models often exhibit certain limitations due to a rapid increase in the number of scenarios to consider, which often prevents their practical application (Alonso-Ayuso et al., 2003). In particular, when risk factors are characterized by seasonality patterns, the number of stages required leads to mathematical models that cannot be solved with reasonable computational resources or within an acceptable amount of time. This challenge, known as the “curse of dimensionality”, underscores the importance of developing efficient heuristics capable of handling

large-scale, complex problems without sacrificing solution quality or computational tractability (de Kok et al., 2018).

A potential approach to address these limitations is through the use of reinforcement learning (RL), which, as evidenced by Boute et al. (2022), has rarely been applied to the SCIM domain. Recently, deep reinforcement learning (DRL) algorithms have attracted increasing attention and have been employed to tackle the SCIM problem, demonstrating their effectiveness and efficiency (Yan et al., 2022). However, several challenges persist (Dulac-Arnold et al., 2021), including: (i) problem-specific properties can be difficult to capture as most DRL algorithms are model-free; (ii) performances of DRL algorithms can be highly sensitive due to the complex task of hyperparameter tuning; and (iii) significant training time is required as DRL algorithms are computationally intensive.

Furthermore, several critical aspects of the SCIM problem have not yet been effectively addressed in the RL context, for example: (i) incorporating a seasonal and stochastic demand, which considerably complicates the determination of actions to be taken; (ii) considering production constraints that make it necessary, for instance, to maintain stocks in advance to prevent myopic behavior; and (iii) addressing fixed

* Corresponding author at: Department of Informatics, Systems, and Communication (DISCo), University of Milano-Bicocca, Viale Sarca, 336, Milan, 20126, Italy.

E-mail addresses: francesco.stranieri@polito.it (F. Stranieri), edoardo.fadda@polito.it (E. Fadda), fabio.stella@unimib.it (F. Stella).

<https://doi.org/10.1016/j.ijpe.2023.109099>

Received 4 May 2023; Received in revised form 20 October 2023; Accepted 13 November 2023

Available online 15 November 2023

0925-5273/© 2023 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

and variable logistic costs (i.e., costs for each vehicle utilized and each unit of item shipped, respectively), which needs proper optimization of production and shipments. Notably, accounting for this final point is crucial not only for economic efficiency but also for reducing the environmental impact of the supply chain and fostering *sustainability*.

Consequently, there is a growing need for more advanced RL approaches that can effectively tackle these challenges and deliver better performance in complex SCIM settings. In an effort to fill these gaps in the existing literature, in this paper we:

- Consider a two-echelon supply chain (i.e., consisting of a factory and multiple distribution warehouses) with a single-item type characterized by seasonal demand, constraints on production and warehouses' capacity, and logistics with fixed and variable costs.
- Design and develop a SCIM software environment capable of modeling all the aforementioned characteristics. We have made the developed SCIM environment accessible as an open-source library on GitHub.¹
- Propose a novel heuristic that combines DRL algorithms and MS stochastic programming to compute an effective solution for the presented SCIM problem.
- Demonstrate the usefulness of the proposed heuristic through a series of numerical experiment settings (e.g., by varying the uncertainty associated with the demand and the number of distribution warehouses).

The rest of the paper is organized as follows: Section 2 provides a literature review on the SCIM problem, including state-of-the-art algorithms used to address it. Section 3 is devoted to mathematically introducing the SCIM problem, while Section 4 presents the solution techniques implemented in this work. Results from numerical experiments are reported in Section 5. Finally, conclusions and potential directions for further research are discussed in Section 6.

2. Literature review

The SCIM problem has been tackled through several approaches due to its sequential and stochastic nature, making it one of the most challenging and significant problems in production research. The most prominent solution approaches include multi-stage stochastic programming (Khouja, 2003; Preusser et al., 2010; Huang et al., 2010), reordering policies (Zipkin, 2000; Grewal et al., 2015), and, more recently, reinforcement learning (Peng et al., 2019; Hubbs et al., 2020; Gijbrecchts et al., 2022; Stranieri and Stella, 2022).

MS optimization employs *scenario trees* to represent uncertainty. Despite some applications when demand is stationary (Khouja, 2003; Huang et al., 2010), its use in settings characterized by seasonal demand has been limited. In fact, MS requires a large number of stages to effectively model seasonality, which leads to an increase in the number of *decision variables*, thus requiring remarkable computation resources to obtain a solution. As a result, much of the existing literature focuses on computationally lighter solution methods, such as reordering policies (Rolf et al., 2022).

More in detail, reordering policies are *decision rules* that determine the number of items (or batches) to produce and ship. One of the earliest rules derived from lot-sizing literature is the *base-stock policy* or (s, S) -policy (Wagner and Whitin, 1958). According to this rule, if the stock quantity falls below s , an order is placed to bring inventory back up to the base-stock level S . A variation of this rule is the so-called (s, Q) -policy, where $Q = S - s$ (Vincent, 1985). Therefore, under an (s, Q) -policy, whenever the stock level falls below s , an order of quantity Q is placed to replenish the inventory.

Reordering policies pave the way for several other rules. In particular, reordering policies are further generalized in RL, which considers

rules encompassing estimations of the future. One of the most common approaches for solving the SCIM problem through RL algorithms is by *Q-learning* (Ravulapati et al., 2004; Chaharsooghi et al., 2008; Sui et al., 2010; Mortazavi et al., 2015). However, upon examining various RL studies, it becomes evident that the implemented Q-tables are typically huge and, thus, unscalable, and this becomes worst when seasonal and stochastic demand is considered. Consequently, since a tabular representation is out of the question, more sophisticated representations may be useful.

One of the most promising techniques that has been successfully applied in several settings is DRL, which combines *deep learning* methods with the classical reinforcement learning paradigm. Despite its success achieved in other domains, to the best of the authors' knowledge, only a few papers have implemented DRL algorithms to address the SCIM problem.

In Hubbs et al. (2020), the authors examine a four-echelon supply chain, employing the Proximal Policy Optimization (PPO) algorithm. The study compares the performance of various operations research methods with a DRL approach in two scenarios, one with and one without backorder costs. Numerical experiments prove that PPO outperforms reordering policies in both scenarios.

Using a supply chain structure with ten warehouses and a normal demand distribution, in Gijbrecchts et al. (2022), the authors conduct two distinct numerical experiments (adapted from Roy et al. (1997)) by applying and tuning the Asynchronous Advantage Actor-Critic (A3C) algorithm. The proposed solution method involves a state-dependent base-stock policy to restrict the action space, with A3C used to select the base-stock level at each timestep. The study shows that A3C achieved performance comparable to state-of-the-art heuristics and other RL algorithms, although its initial hyperparameter tuning remains computationally intensive.

Despite their successes, DRL approaches face several challenges, including a lack of robust convergence properties, strong sensitivity to hyperparameters, high sample complexity, and function approximation errors. These issues can result in poor policies and inaccurate value estimation (Fujimoto et al., 2018; Henderson et al., 2018; Harsha et al., 2021). To deal with these challenges, researchers have attempted to combine DRL techniques with MILP.

One such approach is presented in Bertsekas (2021), which demonstrates that by adding a rough approximation of the value function to the MS objective function, it is possible to enhance the performance of MS significantly.

Another possible solution method is provided by Harsha et al. (2021), where the authors propose RL methodologies that compute actions by solving a MILP problem. Here, the objective function combines an immediate reward with a value function estimated by a trained neural network. Interestingly, due to the simple structure of the neural network and the use of ReLU activation functions, the objective function can be expressed as a linear term, thus leading to a linear objective function. The authors found that this technique outperforms other methods in various realistic settings.

In this work, we adopt a different approach compared to previous studies. Instead of using RL to reduce the myopic behavior of mathematical models, we *decompose* the SCIM problem into two parts. First, we use RL to make production decisions that require considering a long horizon. Second, we employ MS programming for the logistic decisions that need to be made over a shorter horizon. While various techniques exist to address settings with independent and identically distributed (i.i.d.) demand (Yan et al., 2022), our work delves into more complex settings. In particular, we consider a two-echelon supply chain with seasonal demand, constraints on production, limited warehouse capacities, and fixed and variable logistic costs, making our environment more realistic.

In fact, to the best of the authors' knowledge, the only study in the RL literature assuming seasonal demand within a two-echelon supply chain is Peng et al. (2019). Here, a DRL approach based on the Vanilla

¹ <https://github.com/frenkowski/SCIMAI-Gym>.

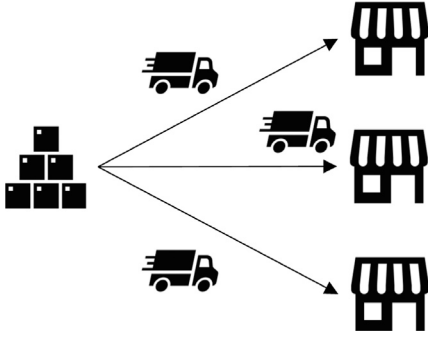


Fig. 1. A two-echelon supply chain consisting of a factory (first echelon) and three distribution warehouses (second echelon).

Policy Gradient (VPG) algorithm is used. The authors evaluate the VPG performance through numerical experiments on three distinct cases. The findings underscore that the VPG algorithm outperforms the (s, Q) -policy employed as a baseline in all three cases. Nevertheless, despite their setting being similar to ours, they do not consider production constraints, warehouse capacities, and fixed and variable logistic costs, making our work the first to explore such a setting.

3. Problem formulation

In this work, we deal with a *divergent supply chain*, i.e., a supply chain characterized by a single participant in the first echelon and multiple participants in the second echelon. More specifically, we investigate a *single-item-type, two-echelon supply chain* consisting of a single factory at the first echelon and a set $\mathcal{J} = \{1, 2, \dots, J\}$ of distribution warehouses at the second echelon, as depicted in Fig. 1. Despite its apparent simple structure, it aptly describes several settings, such as the fuel industry supply chain in which there is a refinery and multiple points of sale or, more generally, two-echelon supply chains where each item type is managed independently.

In detail, we consider episodes of length T (i.e., the time horizon) subdivided into a given number of equally spaced timesteps, implying a supply chain under *periodic review*. At the beginning of each episode, each warehouse $j = 0, 1, \dots, J$ has an initial number of batches $\bar{I}_{j,0}$. At each timestep t , the agent decides the number of batches to produce, x_t , (which must be lower than the factory production limit X^{\max}), paying c_0 for each of them. Batches can either be stored in the factory or shipped to distribution warehouses. In the first case, the agent pays h_0 for each batch and can store up to I_0^{\max} of them. In the second case, the agent ships $z_{j,t}$ batches to the distribution warehouse j . Each distribution warehouse j has a maximum capacity of I_j^{\max} , a storage cost of h_j per batch, and a stock level at timestep t equal to $I_{j,t}$.

We assume *infinite delivery capacity*, meaning we can ship any quantity of stocks at the same cost. The logistic cost comprises a fixed price p_j paid for each vehicle used to transport batches to warehouse j , irrespective of the truckload, and a variable price l_j dependent on the number of batches shipped. Defining V_j as the capacity of the vehicles going to the distribution warehouse j , $y_{j,t} = \lceil \frac{z_{j,t}}{V_j} \rceil$ represents the associated number of vehicles used for the shipment. Clearly, from a logistic perspective, the goal of the agent is to achieve *full-truckload shipping*. In fact, by fully utilizing the truck's capacity, the agent can optimize vehicle resources, leading to lower shipping costs and increased overall efficiency in the supply chain. This approach helps minimize the number of partially filled trucks, reducing fuel consumption and emissions, as well as the number of expeditions required to satisfy demand. Consequently, it contributes to a more cost-effective and environmentally friendly SCIM strategy.

We consider a *seasonal and stochastic demand*, $d_{j,t}$, that realizes, for each timestep t , at distribution warehouse j . Unsatisfied demand is

backordered, meaning that if the demand cannot be satisfied, a penalty cost b_j is incurred until the specific distribution warehouse j is able to satisfy it. Finally, we assume no lead times for production and logistics; thus, inventories are used to make a *seasonal stock* (Brandimarte and Zotteri, 2007). A summary of the notation is provided in Table 1.

The *dynamics of the system* are graphically represented in Fig. 2 and described by the following sequence of events:

1. Starting from the current state of the environment, the agent determines the number of batches to produce, x_t , and ship, $z_{j,t}$, to each distribution warehouse $j = 1, \dots, J$.
2. Each distribution warehouse $j = 1, \dots, J$ receives the sent batches of stocks, $z_{j,t}$.
3. Demand $d_{j,t}$ at each distribution warehouse $j = 1, \dots, J$ is either satisfied or backordered.
4. The *per-step cost*, C_t , is calculated according to the following formula:

$$C_t = \underbrace{c_0 \cdot x_t}_{\text{production costs}} + \underbrace{\sum_{j=1}^J l_j \cdot z_{j,t}}_{\text{variable logistic costs}} + \underbrace{\sum_{j=1}^J p_j \cdot y_{j,t}}_{\text{fixed logistic costs}} + \underbrace{\sum_{j=0}^J h_j \cdot \max[I_{j,t}, 0]}_{\text{storage costs}} - \underbrace{\sum_{j=0}^J b_j \cdot \min[I_{j,t}, 0]}_{\text{backorder costs}} \quad (1)$$

where the first term represents production costs, the second and the third consist of variable and fixed logistic costs, respectively, the fourth denotes storage costs, and the last one quantifies backorder costs (which are introduced with a minus sign because stock levels would be negative in the eventuality of unsatisfied demand).

5. The state related to the next timestep is determined, transferring surplus stocks or unsatisfied demands. Formally, the evolution of the inventories is defined as follows:

$$\begin{cases} I_{0,t+1} = \min[(I_{0,t} + x_t), I_0^{\max}] - \sum_{j=1}^J z_{j,t} \\ I_{1,t+1} = \min[(I_{1,t} + z_{1,t-1}), I_1^{\max}] - d_{1,t} \\ \vdots \\ I_{J,t+1} = \min[(I_{J,t} + z_{J,t-1}), I_J^{\max}] - d_{J,t} \end{cases} \quad (2)$$

This implies that at the beginning of the timestep $t+1$, the factory's stocks are equal to the sum of the stocks at timestep t and the production during the same period, minus the batches shipped during timestep t . Similarly, the distribution warehouses' stocks at timestep $t+1$ are equal to the stocks at timestep t , plus the batches received from the factory during the same period, minus the demand at timestep t . It is worth mentioning that the environment does not allow storing a number of batches exceeding the storage capacity constraints, automatically discarding all batches that, if accepted, would violate these constraints.

4. Solution methods

In this section, we present the techniques used to solve the problem described in Section 3. Section 4.1 offers a brief introduction to deep reinforcement learning, while Section 4.2 defines the multi-stage stochastic programming model. Finally, Section 4.3 describes the proposed heuristic, which combines DRL and MS optimization.

4.1. Deep reinforcement learning

RL adopts the Markov decision process (MDP) framework to represent the interactions between a *learning agent* and an *environment*. As shown in Fig. 3, at each timestep t , the agent observes the current

Table 1
The considered SCIM notation with relative explanation (and units of measure).

Notation	Explanation	Notation	Explanation
T	Time horizon	I_j^{\max}	Storage capacity (batches)
J	Number of distribution warehouses	h_j	Storage cost (per batch)
\mathcal{J}	Set of distribution warehouses	p_j	Logistic cost fixed (per vehicle)
$d_{j,t}$	Demand (batches)	l_j	Logistic cost variable (per batch)
x_t	Production (batches)	$z_{j,t}$	Shipping (batches)
c_0	Production cost (per batch)	V_j	Vehicles capacities (batches)
X^{\max}	Maximum production (batches)	$y_{j,t}$	Number of vehicles
$I_{j,t}$	Stock level (batches)	b_j	Backorder cost (per batch)

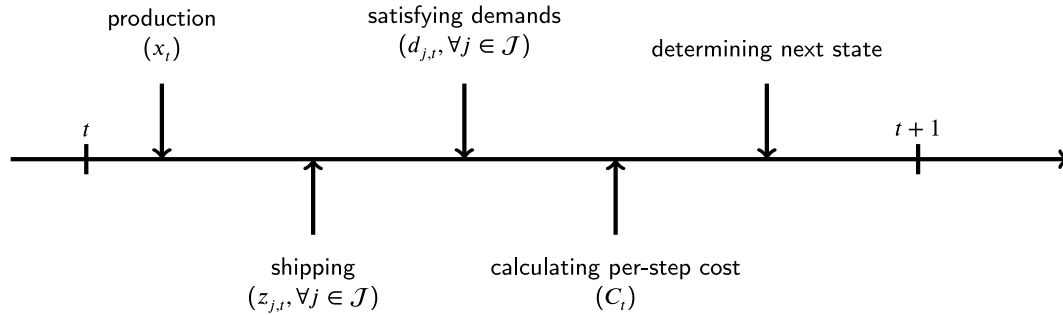


Fig. 2. Representation of the dynamics of the SCIM system considered.

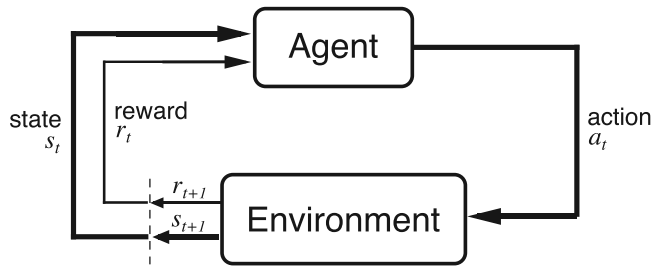


Fig. 3. Agent-environment interface in an MDP (Sutton and Barto, 2018).

state of the environment, s_t , selects an action, a_t , and obtains a reward, $R_{t+1} \in \mathbb{R}$; afterward, the environment transitions into a new state, s_{t+1} . The goal of RL is to find an *optimal policy* – a function that maps the states of the environment to a set of actions – that maximizes the *expected discounted return*, $\sum_{k=t+1}^T \gamma^{k-t-1} R_k$, where $0 \leq \gamma \leq 1$ represents the discount rate (Powell, 2011). In the SCIM problem, the objective is to minimize the total cost, therefore we define the reward as the negative cost, i.e., $R_t = -C_t$.

DRL combines RL with deep learning, offering the potential to scale to previously intractable decision-making problems. In detail, DRL is rooted in neural networks, which are universal approximators capable of expressing an approximation of highly nonlinear functions. The specific DRL algorithm we used belongs to *policy-based methods*, which learn a parameterized and stochastic policy to select actions directly (in contrast to value-based methods like Q-learning (Sutton and Barto, 2018)). To formally address and solve the SCIM problem within the context of an MDP, we thus need to define the state vector, the action space, and the reward function relating to the SCIM problem assumed.

The *state vector* includes all current stock levels (for the factory and each distribution warehouse) and the last τ demand values; this provides the agent with limited knowledge of demand history, which, in turn, allows for a basic understanding of its fluctuations (similar to what was initially proposed by Kemmer et al., 2018). In mathematical terms, we have:

$$s_t = (I_{0,t}, \dots, I_{J,t}, \mathbf{d}_{t-1}, \dots, \mathbf{d}_{t-\tau}), \quad (3)$$

where $\mathbf{d}_t = (d_{0,t}, \dots, d_{J,t})$. The *state's updating rule* is dictated by Eq. (2) for the inventory, while the values of the demands are updated based on

the current timestep. It is worth noting that during timestep t , the actual demand \mathbf{d}_t is only known after decisions regarding production and shipping have been made; this ensures that the agent can benefit from learning the demand pattern, allowing for the integration of *demand forecasting* directly into the policy. Further, by observing and learning from the demand history, the agent can develop a more effective decision-making process, anticipating future demand fluctuations and adjusting its actions accordingly (Stranieri and Stella, 2022).

Concerning the *action space*, it contains information about production and shipping controls:

$$\mathbf{a}_t = (x_t, z_{1,t}, \dots, z_{J,t}). \quad (4)$$

The considered agent uses a *continuous action space* – wherein the neural network directly generates the action values – since it scales better than a discrete action space and can be applied to wider action spaces (Vanvuchelen and Boute, 2022). As a result, the agent can specify the factory's production level and the number of batches to ship to each distribution warehouse.

To guarantee that the actions generated by the neural network belong to the feasible action space, we adopted a continuous action space based on independent bounds (Stranieri and Stella, 2022). This involves: (i) setting the lower bound for each action value to zero; (ii) setting the upper bound for the factory to its maximum production level (i.e., $0 \leq x_t \leq X^{\max}$); and (iii) setting the upper bound for each warehouse to its corresponding storage capacity (i.e., $0 \leq z_{j,t} \leq I_j^{\max}, \forall j \in \mathcal{J}$). It is worth noticing that if the agent produces or ships a number of batches exceeding the free storage availability (for example, due to stocks preserved in the warehouses), the environment directly discards those in excess, according to Eq. (2). This results in a cost, which implicitly penalizes the agent, thus encouraging it to learn a policy that prevents overproduction and excessive shipping. Additionally, when factory stocks are insufficient to meet orders from all distribution warehouses, we employ the allocation rule described in Stranieri et al. (2023) to ensure a fair distribution of batches among the distribution warehouses while maintaining factory stocks non-negative.

Finally, the *reward function* is defined by means of Eq. (1).

4.2. Multi-stage stochastic programming

In multi-stage stochastic programming, the uncertainty associated with demand is represented through a *scenario tree*, as depicted in

Fig. 4. Each layer of the tree corresponds to a stage and models the available information. In the SCIM problem, because new information (i.e., demand realization) becomes available at each time step, the concepts of stage and time steps are equivalent.

The leftmost node (labeled as node 0 in Fig. 4) represents the current state of the environment where we determine the *first-stage decisions*, each associated with an immediate and definite per-step cost. These decisions influence the second-stage ones, made after observing the realized demand (as illustrated by nodes 1 and 2 in Fig. 4). Then, in the subsequent stages of the scenario tree, the process is repeated (Birge and Louveaux, 2011). Notably, decisions made after the first stage can be viewed as *contingent plans* based on future demands realizations.

To this end, we let:

- \mathcal{N} be the set of nodes of the scenario tree, $\mathcal{N}^+ = \mathcal{N} \setminus \{0\}$, with 0 being the root node.
- $p(n)$ be the *parent* of node $n \in \mathcal{N}^+$; for example, in Fig. 4, $p(1) = p(2) = 0$, $p(3) = p(4) = 1$, ...
- $\pi^{[n]}$ be the *unconditional probability* of node n (i.e., the likelihood of demand realization in node n expressed considering the information available from the root node) (Brandimarte, 2011); for example, in Fig. 4, we have $\pi^{[0]} = 1$, $\pi^{[1]} = \pi^{[2]} = \frac{1}{2}$, $\pi^{[3]} = \pi^{[4]} = \pi^{[5]} = \pi^{[6]} = \frac{1}{4}$, ...
- $d_j^{[n]}$ be the item demand from distribution warehouse j at node $n \in \mathcal{N}$.

In the MS model, given that the decision variables are contingent on the node of the scenario tree (which includes all available information), we replace the subscript \cdot with the superscript $^{[n]}$ for all variables. The superscript $^{[n]}$ denotes a specific node of the scenario tree and is consequently associated with a single stage and a single time step. For example, x_t in Table 1 becomes $x_t^{[n]}$, representing the quantity produced at node n . It is essential to note that our primary interest lies in the variables calculated in the first stage, in particular, the quantity to produce ($x^{[0]}$), the number of batches to ship ($z_j^{[0]}, \forall j \in \mathcal{J}$), and the number of vehicles to be employed ($y_j^{[0]}, \forall j \in \mathcal{J}$).

The MS optimization model can thus be defined as follows:

$$\begin{aligned} \min_{\substack{x^{[n]}, y_j^{[n]}, z_j^{[n]}, I_j^{[n]} \\ \forall n \in \mathcal{N}, \forall j \in \mathcal{J}}} \quad & \sum_{n \in \mathcal{N}} \pi^{[n]} \left[c_0 x^{[n]} + \sum_{j=1}^J l_j z_j^{[n]} \right. \\ & + \sum_{j=1}^J p_j y_j^{[n]} + \sum_{j=0}^J h_j I_j^{+[n]} \\ & \left. + \sum_{j=0}^J b_j I_j^{-[n]} \right] \end{aligned} \quad (5)$$

$$\text{s.t.} \quad I_0^{[n]} = I_0^{[p(n)]} + x^{[n]} - \sum_{j=1}^J z_j^{[n]} \quad \forall n \in \mathcal{N}^+ \quad (6)$$

$$I_j^{[n]} = I_j^{[p(n)]} + z_j^{[p(n)]} - d_j^{[n]} \quad \forall n \in \mathcal{N}^+, \forall j \in \mathcal{J} \quad (7)$$

$$I_j^{[n]} = I_j^{+[n]} - I_j^{-[n]} \quad \forall n \in \mathcal{N}, \forall j \in \mathcal{J} \cup \{0\} \quad (8)$$

$$I_j^{[0]} = \bar{I}_j^{[0]} \quad \forall j \in \mathcal{J} \cup \{0\} \quad (9)$$

$$y_j^{[n]} \geq \frac{z_j^{[n]}}{V_j} \quad \forall n \in \mathcal{N}, \forall j \in \mathcal{J} \quad (10)$$

$$x^{[n]} \in \{0, \dots, X^{\max}\} \quad \forall n \in \mathcal{N} \quad (11)$$

$$z_j^{[n]} \in \mathbb{Z}_+, y_j^{[n]} \in \mathbb{Z}_+ \quad \forall n \in \mathcal{N}, \forall j \in \mathcal{J} \quad (12)$$

$$I_j^{[n]} \in \mathbb{Z}, I_j^{+[n]} \in \{0, \dots, I_j^{\max}\}, \quad (13)$$

$$I_j^{-[n]} \in \mathbb{Z}_+ \quad \forall n \in \mathcal{N}, \forall j \in \mathcal{J} \cup \{0\}.$$

The objective function in Eq. (5) represents the expected total costs, expressed as the sum of production, logistics, storage, and backorder

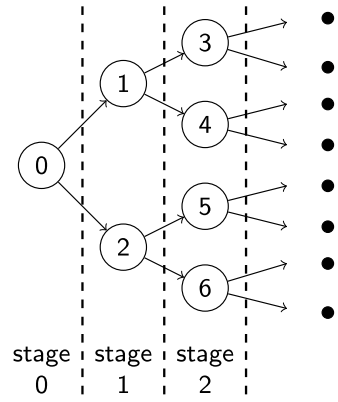


Fig. 4. Scenario tree representation where each node represents a possible demand realization.

costs. Constraints (6)–(7) dictate the evolution of the stocks in the factory and distribution warehouses. These constraints are analogous to Eq. (2) but contextualized to MS stochastic programming. Moreover, constraints (8) define the relationship between the variable $I^{[n]}$, the inventory $I^{+[n]}$, and the backorder $I^{-[n]}$. These latter two are variables used to linearize the max and min operators in Eq. (1). Lastly, constraints (9) establish the initial stock condition, constraints (10) determine the number of vehicles used for shipping, and constraints (11)–(13) describe the domains to which the variables belong. It is important to note that model (5)–(13) implicitly enforces the *non-anticipative constraints*, meaning that the decision-maker cannot exploit future information (Brandimarte, 2011). For example, in Fig. 4, at node 2, the demand realizations for both nodes 5 and 6 are possible. By solving this MS optimization model, the SCIM problem can be optimized to *minimize total costs* while trying to satisfy the overall demand from the distribution warehouses.

4.3. Deep reinforcement learning-based decomposition

The proposed heuristic combines the main points of strength of both DRL and MS programming, leveraging their complementary characteristics to address their respective weaknesses.

The principal advantage of MS programming lies in its model-based problem formulation, where both the objective function and the constraints are *explicitly defined* within the model; this is particularly effective when the model accurately represents the problem, as in the considered context. Indeed, the mathematical model adeptly captures the computation of inventories evolution and backlogs, the number of vehicles to use, and their consequent impact on the objective function. By contrast, the main drawback of MS programming stands in its potential need for multiple stages to properly represent a stochastic and seasonal demand, thus leading to challenging problems that, in the worst case, cannot be solved by an exact solver within a reasonable amount of time. Specifically, the bottleneck in MS programming is the number of scenarios required to approximate the *out-of-sample cost* of the solution when complex demands are considered. Seasonal demand is a prime example where several stages – and thus several scenarios – are needed to characterize the evolving demand distribution.

In general, given J distribution warehouses, N possible demand realizations for each stage, and T stages (assuming the number of stages equals the time horizon), we end up with N^T scenarios and $(5J+2) \cdot N^T$ variables. Even small values of J , N , and T prevent an exact solver from computing the optimal solution of the model. Furthermore, in the SCIM problem, the *risk mitigation strategy* involves producing and storing batches in warehouses, thereby paying relative storage costs. Hence, inventory decisions require the longest look ahead. Conversely, seasonality is not a major issue for DRL since it adapts its strategy to

random demand based on the *policy learned* during the training phase. Although the training phase for complex demands can take time, it is only required once. Thus, in a testing environment, the solution is nearly immediate. Nevertheless, being a model-free method, DRL suffers from slow convergence, a weakness that becomes even more marked as the number of constraints increases, as it happens in our current setting.

The complementary strengths and weaknesses of MS programming and DRL suggest a potential synergy if combined to develop a novel heuristic, exploiting the policy provided by DRL for decisions requiring a longer time horizon and using MS to determine all other variables that need a shorter time horizon.

More in detail, we consider two distinct agents at play, one driven by DRL and the other by MS programming. In the following, we will employ the MDP notation \cdot_t to refer to the variables associated with the DRL agent while using the notation $\cdot^{[n]}$ for those tied to the MS agent. Thus, for instance, x_t denotes the MDP variable that describes the production quantity at timestep t , whereas $x^{[0]}$ pertains to the MS model (5)–(13), which will be solved with a rolling horizon approach to determine the production quantity at timestep t .

Initially, we train the DRL agent over a set of episodes; then, we use the trained DRL policy to select the number of batches to produce, thereby setting the value x_t . Identifying an optimal value for this decision variable requires deep foresight; moreover, any value within the range $[0, X^{\max}]$ yields a feasible solution, eliminating the need for the DRL agent to learn complex constraints. Once the decision about the number of batches to produce has been made, the second agent implements an MS model to handle the logistics decisions. Specifically, it solves model (5)–(13) by fixing the $x^{[0]}$ variable with the DRL's derived solution.

Since the production decisions now depend only on the DRL agent, the MS agent no longer needs to consider an extensive number of stages. Thus, we reduce the number of stages to two, leading to a significantly more manageable model. However, the number of scenarios within the second stage might still lead to computational issues. To counteract this, we reduce the number of scenarios using Monte Carlo techniques, such as *moment matching generation* or *importance sampling* (Brandimarte, 2014). Since all the second-stage variables are used to account for the future, we relax them to be continuous, and we only constrain the first-stage variables $y_j^{[0]}$ to be integers. This strategy has been successfully employed in *fix-and-relax heuristics* (Brandimarte, 2006). By adopting this approach, despite the introduction of approximation errors at future stages, we can guarantee the quick achievement of a feasible solution.

The pseudocode summarizing the heuristic is outlined in Algorithm 1. In the following sections, we will refer to this heuristic as deep reinforcement learning-based decomposition (DRLBD).

Algorithm 1 Deep Reinforcement Learning-Based Decomposition (DRLBD) algorithm.

```

1: Train a DRL agent
2: for  $t = 1, \dots, T$  do
3:   Observe the current state of the environment,  $s_t$ 
4:   Determine the production level  $x_t$  using the DRL agent
5:   Generate a scenario tree
6:   Set  $x^{[0]} \leftarrow x_t$  in model (5)–(13)
7:   Relax variables  $y_j^{[n]} \in \mathbb{R}_+$ ,  $\forall j = 1, \dots, J$ ,  $n \neq 0$ 
8:   Solve model (5)–(13) using the MS agent
9:   Set  $z_{j,t} \leftarrow z_{j,t}^{[0]}$ ,  $\forall j = 1, \dots, J$ 
10:  Produce  $x_t$  and ship  $z_{j,t}$ ,  $\forall j = 1, \dots, J$ 
11:  Satisfy demand  $d_{j,t}$ ,  $\forall j = 1, \dots, J$ 
12:  Calculate per-step cost  $C_t$  and determine next state  $s_{t+1}$ 
13: end for

```

5. Numerical experiments

This section presents the numerical experiments designed to evaluate the performance of the proposed heuristic algorithm. All experiments were executed on a machine equipped with an Apple M2 Pro chip with 10 cores and 16 GB of RAM. We developed the code using Python 3.10. Specifically, we used: (i) the OpenAI Gym APIs (Brockman et al., 2016) to implement the SCIM environment; (ii) the Ray Python library (Moritz et al., 2017) for importing the DRL algorithms; (iii) Gurobi version 10.0 (via its Python APIs) (Gurobi Optimization, LLC, 2023) to solve the MS optimization model.

We organize the numerical experiments into *two different settings*. The first setting is elementary and aims to compare the methods under consideration with the derived optimal solution. The second setting involves more challenging experiments to evaluate in-depth the performance of the proposed heuristic in scenarios where an optimal solution cannot be computed within a reasonable amount of time. The specific values of the parameters that define the SCIM environment for both small and large settings are reported in Appendix A. Due to the lack of real-world data or benchmark instances, our experimental plan relies on *synthetic and realistic data*. However, by providing open access to our code, we aim to offer a reference point for future research, thereby addressing this limitation. While the data used in Section 5.1 ensure a specific problem structure but lacks realism, the data in Section 5.2 has been validated in collaboration with Bristol-Myers Squibb² to simulate a realistic supply chain.

Following Kemmer et al. (2018), and Peng et al. (2019), we define the seasonal demand for warehouse j as:

$$d_{j,t} = \left\lfloor D_j \left(1 + \sin \left(\frac{2\pi(t - \phi_j)}{P_j} \right) \right) \right\rfloor + \epsilon_j, \quad \forall j = 1, \dots, J, \forall t = 1, \dots, T, \quad (14)$$

where $\lfloor \cdot \rfloor$ represents the floor function, D_j denotes the amplitude of the reference demand value (set to exceed the maximal production X^{\max}), ϕ_j is the phase, P_j represents the period (set as a fraction of T), and ϵ_j indicates a *random noise* which will be defined differently in each experimental setting. Fig. 5 provides an empirical example of a 95% confidence interval where $D_j = 5$, $P_j = 5$ over a time horizon of $T = 7$ timesteps, and $\phi_j = 0$, with ϵ_j as in Eq. (16).

If $\phi_j \neq 0$ and $P_j \neq P_k, \forall j \neq k \in J$, demands manifest peaks at different timesteps. Accordingly, summing the demands across all warehouses will diminish the impact of these peaks, enabling the agent to make good decisions with a short look-ahead. Given these insights, we consider $\phi_j = 0, \forall j \in J$, and $P_j = P_k, \forall j, k \in J$; this represents the most challenging scenario, characterized by *complete demands overlapping* that results in pronounced demand peaks, thus necessitating a wise production and stocking strategy. Such a scenario is particularly plausible when warehouses are located in homogeneous regions that follow equivalent demand patterns. Examples include the paper notebook industry, where sales synchronize with the academic calendar (consistent at a national level), or the food industry producing items linked to specific times of the year (such as Christmas cakes). In fact, production might only occur in just a few periods of the year in these supply chains.

For the sake of simplicity, we assume that random noises ϵ_j are i.i.d. across all distribution warehouses j and that all D_j values are equal. As a result, we will omit the subscript j from ϵ_j , P_j , and D_j in subsequent sections. Additionally, we assume that when production capacity is used strategically, all demands can be satisfied; mathematically, this means that:

$$\frac{P \cdot X^{\max}}{J} \geq \sum_{t=0}^P \left\lfloor D \left(1 + \sin \left(\frac{2\pi(t - \phi_j)}{P} \right) \right) \right\rfloor + \bar{\epsilon}, \quad (15)$$

² <https://www.bms.com/>.

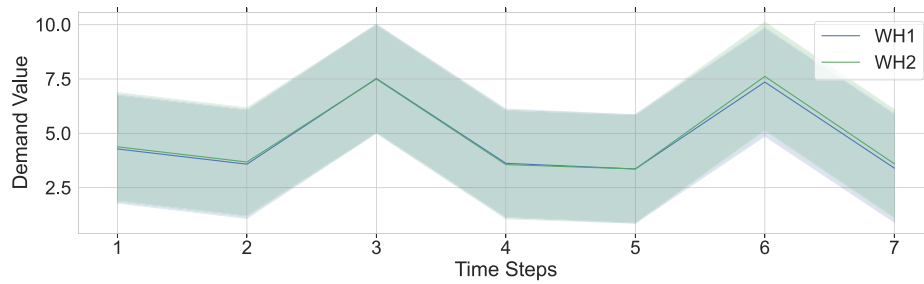


Fig. 5. 95% confidence interval computed over 250 realizations of the seasonal demand with two distribution warehouses (i.e., WH1 and WH2), $D_j = 5$, $P_j = 5$, $T = 7$, $\phi_j = 0$, and ϵ as in Eq. (16).

where $\bar{\epsilon}$ represents the average demand noise from a single distribution warehouse.

For performance comparisons, we calculate the total costs as $\sum_{t=1}^T C_t$ for each experiment, and we replicate this process 250 times. As benchmark techniques, we exploit the PPO algorithm (Schulman et al., 2017), given its demonstrated superiority in comparable settings over other state-of-the-art DRL algorithms, such as VPG and A3C (Stranieri and Stella, 2022; Stranieri et al., 2023). For the same reasons, DRLBD also adopts PPO as its foundation algorithm. The hyperparameters for DRLBD and PPO are detailed in Appendix B.

5.1. Small settings

In the first context, we examine experiments attributable to a *small setting* involving two distribution warehouses with demand parameters $D = 5$ and $P = 5$, spanning a time horizon of $T = 7$ timesteps where each timestep conceptually represents a day. The primary goal of this setting is to compute the optimality gap of the proposed heuristic rather than evaluate its performance in a realistic context.

In detail, two experiments are conducted within this setting: the first considers a noise ϵ distributed according to a Bernoulli distribution with a probability of 0.5, while the second experiment employs a different distribution for the noise ϵ , defined as follows:

$$\epsilon = \begin{cases} 0 & \text{with probability 0.5} \\ 5 & \text{with probability 0.5.} \end{cases} \quad (16)$$

We select these two distributions because, for a specific number of batches, they model to either have an increased demand over a short number of timesteps or none at all. Furthermore, an optimal solution to the problem can be easily computed using these distributions.

One viable strategy would involve value iteration or policy iteration (Brandimarte, 2021). However, these methods require an excessive amount of time to reach convergence. As a result, we opt to directly solve the MS problem as expressed in model (5)–(13). In fact, since ϵ can take only four distinct values (i.e., two possible outcomes for each of the two distribution warehouses), considering T stages gives rise to 4^T different scenarios. For small values of T , this leads to a mathematical model that can be solved by an exact solver, thereby providing the *optimal solution* (in that case, the scenarios tree precisely describes every possible demand realization with its exact probability).

It is worth noting that there is no advantage in considering a number of stages exceeding the demand period P since, according to Eq. (15), decisions made during the first stage have a *limited influence* on the future. Although not detailed here due to their irrelevance to the scope of the paper, computational experiments suggest that considering only four stages is sufficient to obtain a practically equal solution derived from a model considering all T stages. Thus, the resulting MS stochastic problem considers just ($4^4 =$) 256 scenarios and can be solved by off-the-shelves solvers like Gurobi.

Before delving into a comparison of the proposed heuristic, it is essential to assess the impact of *initial conditions*. To this end, we investigate the cost incurred per timestep starting from empty inventories

over a period of $T = 21$ timesteps. The results for the MS stochastic model are reported in Fig. 6. From both Figs. 6(a) and 6(b), it is evident that there is almost no *transient phase*, as evidenced by the periodicity depicted in the two plots.

To assess the performance of the proposed heuristic, we benchmark it against the PPO algorithm, the (s, Q)-policy (hereafter referred to as sQ), and the *expected value problem* (EVP). Specifically, when employing EVP, we solve the model (5)–(13) by replacing each random variable with its expected value; this leads to a deterministic problem that exact solvers can handle given that the scenario tree depicted in Fig. 4 is reduced to a single chain of length T .

We present the *percentage optimality gaps* derived from simulations in Table 2. Within the table, we also report the performance of the *perfect information* (PI) model. This model is obtained by solving the model (5)–(13) and substituting each random variable with its future realization. Analogous to EVP, this results in a scenario tree comprising T nodes, which can be addressed using exact solvers. It is important to highlight that the PI model assumes complete knowledge of future demand outcomes, making it not implementable in a real-world deployment. Nevertheless, its introduction is helpful for quantifying the inherent value of perfect information regarding the future (Birge and Louveaux, 2011).

As the reader can observe, EVP underperforms in both settings, reporting costs that are 116% and 198% worse than the average MS solution, respectively. Trailing behind EVP, there is sQ, which has an optimality gap of 64% and 47%. Such performance can be attributed to its static nature, which often leads to unsatisfactory solutions, especially in scenarios with seasonal demand. Interestingly, sQ emerges as the only method where the optimality gaps diminish as the variance associated with the noise increases; this observation can be rationalized by recalling that, in the i.i.d. demand context, the (s, Q)-policy is intrinsically optimal (Wagner and Whitin, 1958). Moreover, it is noteworthy that as the noise variance increases, the impact of seasonality on performance diminishes.

In comparison, the PPO algorithm achieves better optimality gaps than sQ, reporting gaps of 24% and 35%, respectively, and attesting to the improved performance of dynamic rules over static ones. Meanwhile, DRLBD reaches an optimality gap of only 6% and 8%, consolidating its position as the most effective approach. Its performance, combined with a narrow standard deviation, reveals that our proposed heuristic consistently earns the lowest optimality gap. In the authors' opinion, these considerable improvements can be mainly attributable to the enhanced logistics management promoted by the MS model.

Lastly, the PI model, as intended, manifests negative optimality gaps, a direct consequence of its ability to access future knowledge. Furthermore, as it is intuitive to guess, PI excels particularly in settings characterized by higher demand variance, where a precise foreknowledge of upcoming scenarios becomes an invaluable asset.

While average values and standard deviations provide some insight, they do not fully describe the actual distribution of costs. Therefore, Fig. 7 delves deeper, presenting the average optimality gaps through histograms of *frequency distributions*. Given that the primary goal of

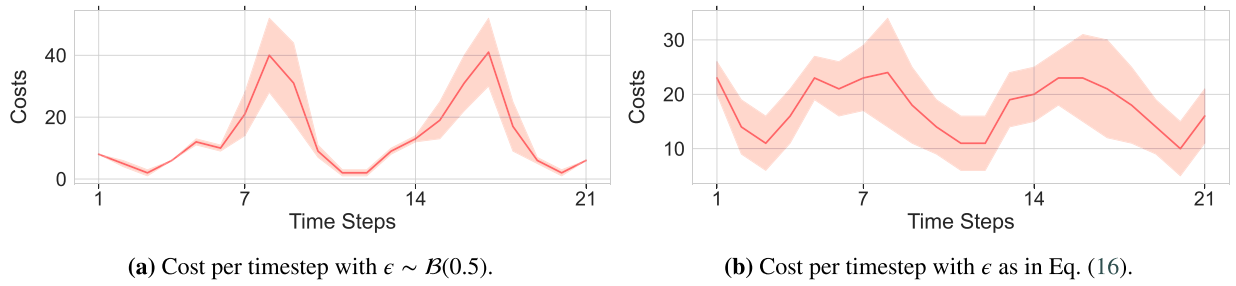


Fig. 6. Cost per timestep over 250 episodes, as computed by the multi-stage programming model. The X -axis represents the timesteps, while the Y -axis displays the per-step cost. From both Figs. 6(a) and 6(b), which refers to the experiments of the small settings (but with $T = 21$), it is possible to observe the periodic behavior of the costs.

Table 2

Average opt-gap (expressed as a percentage) over 250 episodes with respect to the multi-stage programming model. The standard deviation is provided in round brackets. The lower the value, the better the algorithm.

a Average opt-gap (as a percentage) with $\epsilon \sim B(0.5)$.		b Average opt-gap (as a percentage) with ϵ as in Eq. (16).	
Algorithm	Opt-gap [%]	Algorithm	Opt-gap [%]
DRLBD	6.10 (2)	DRLBD	8.66 (3)
PPO	24.67 (9)	PPO	35.66 (9)
sQ	64.09 (30)	sQ	47.71 (12)
EVP	116.95 (59)	EVP	198.62 (160)
PI	-7.79 (3)	PI	-33.63 (9)

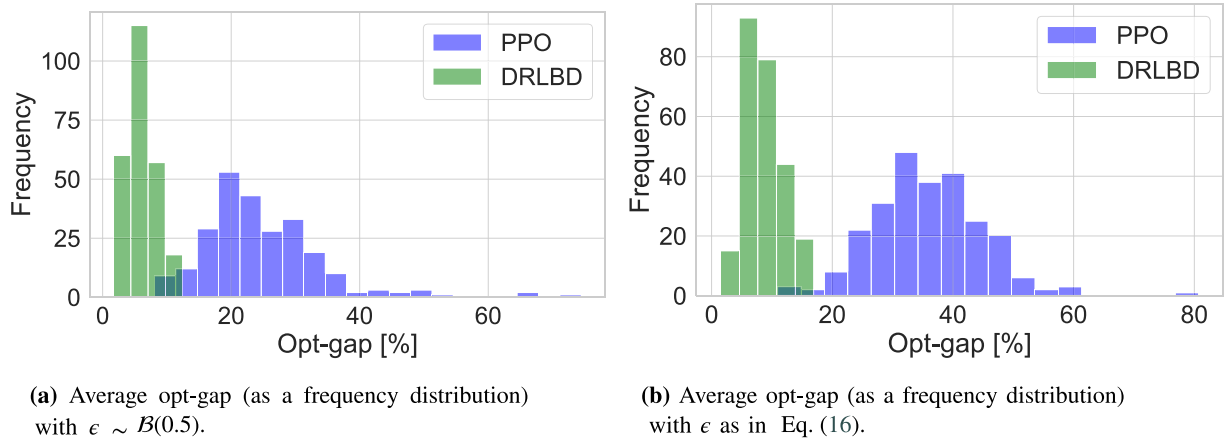


Fig. 7. Average opt-gap (expressed as a frequency distribution) over 250 episodes in comparison to the multi-stage programming model.

this small experiment setting is to evaluate the optimality gap of the proposed methodologies, we focus exclusively on the gaps rather than the absolute cost values. Moreover, our analysis considers only PPO and DRLBD, as they emerged as the most performing techniques.

From Figs. 7(a) and 7(b), it is evident that the cost interval for DRLBD is considerably more confined than that of PPO. In particular, the poorest results for DRLBD – approximately 10% for the first experiment and 15% for the second – are significantly better than those achieved by PPO, which hover around 70% and 80% for the first and second experiments, respectively. Moreover, DRLBD reaches values close to optimality in both experiments, whereas PPO consistently achieves optimality gaps greater than the average values achieved by DRLBD, thus suggesting that DRLBD may offer a more robust and efficient solution than PPO.

Regarding *computational times* for the different techniques, Table 3 contains the training times (in minutes) and the time needed to execute a single episode, referred to as testing times (in milliseconds). Training times for MS, EVP, and PI are omitted from the table, as these methods do not necessitate performing any training phase, while DRLBD has the same training time as PPO since they share the same underlying DRL agent. In this first setting, the computational times are particularly fast. Specifically, all training times require less than 3 min, and all testing

executions are completed in less than a second. However, certain methods diverge from the rest, i.e., MS, which needs to solve a MILP with a considerable number of variables, and EVP, which needs to consider the full-time horizon.

5.2. Large settings

In the second context, we consider experiments within a *large setting* involving 5 or 10 distribution warehouses (i.e., $J = 5$ or $J = 10$).

The demand parameters of Eq. (14) are $D = 2$ and $P = 6$ over a time horizon of one year ($T = 12$), such that each timestep ideally represents a *month*. We consider the noise ϵ of the demand to be a random variable distributed according to a negative binomial distribution with parameters $r = 3$ and $p = 0.7$ (Agrawal and Smith, 1996; Aviv and Federgruen, 1997). Due to the size of the experiments and the demand distribution, in this setting, we do not exploit an MS benchmark since the number of scenarios required to approximate the out-of-sample cost leads to a model that runs *out of memory*. Therefore, we report the results by computing the average gap achieved for each method with respect to the performances reached by PI.

Accordingly, Table 4 presents the results of the comparison among the different benchmark techniques in terms of the expected value

Table 3

Average training and testing times for the two experiments conducted under small settings (i.e., with $\epsilon \sim B(0.5)$ and ϵ as in Eq. (16), respectively) with their standard deviation (in brackets).

Algorithm	Small setting (with $\epsilon \sim B(0.5)$)		Small setting (with ϵ as in Eq. (16))	
	Training time [min]	Testing time [ms]	Training time [min]	Testing time [ms]
DRLBD	2.41 (0.01)	12 (2.01)	2.42 (0.01)	12 (2.01)
PPO	2.41 (0.01)	8 (15.70)	2.42 (0.01)	8 (3.39)
sQ	2.40 (0.01)	5 (0.61)	2.36 (0.01)	5 (0.56)
MS	–	279 (30.30)	–	298 (30.90)
EVP	–	49 (0.52)	–	39 (14.50)
PI	–	7 (0.83)	–	7 (1.18)

Table 4

Average EVPI-gap (expressed as a percentage) over 250 episodes concerning the expected value of perfect information model. The standard deviation is provided in round brackets. The lower the value, the better the algorithm.

a Average EVPI-gap (as a percentage) with $J = 5$ distribution warehouses.		b Average EVPI-gap (as a percentage) with $J = 10$ distribution warehouses.	
Algorithm	EVPI-gap [%]	Algorithm	EVPI-gap [%]
DRLBD	98.08 (21)	DRLBD	105.08 (18)
PPO	132.31 (32)	PPO	139.93 (26)
sQ	145.54 (34)	sQ	188.71 (41)
EVP	548.87 (161)	EVP	577.43 (124)

of perfect information (EVPI). By analyzing the values, it is evident that EVP underperforms, corroborating the conclusion that addressing uncertainty using ad-hoc methods is the appropriate approach. Indeed, the costs incurred by EVP are more than five times higher than those of DRLBD. These poor performances are mainly due to the inability of EVP to produce a sufficiently high amount of items during the initial timesteps of the considered time horizon. As a result, it suffers an initial backlog that becomes impossible to satisfy during the seasonal peak, leading to even higher backlogs. The second underperforming method is sQ which still outperforms EVP since it adopts a massive Q value to prevent backorder accumulation, unlike EVP.

As in the previous experiments, PPO achieves better results than sQ by leveraging its ability to develop a dynamic decision rule proper for seasonal behavior. Finally, DRLBD significantly outperforms all other techniques, showing its superiority compared to the inability of PPO to make advantageous logistic decisions, which, in turn, grows logistic and storage costs.

It is worth noting that all the gaps increase as the number of warehouses grows since it leads to a more challenging problem. Among all methods, sQ and EVP are most impacted by the shift from $J = 5$ to $J = 10$, with performance reductions of 43.17% and 28.56%, respectively. This drop also involves DRLBD and PPO, which decrease their performance solely by 7.00% and 7.62%, respectively, thus proving the stability of the proposed heuristic when confronted with an increasing number of distribution warehouses.

To gain a better understanding of the *cost distribution*, we present it in Fig. 8, expressing the cost in thousands of euros (k€). Both PPO and DRLBD exhibit a similar distribution in both experiments. However, PPO achieves higher average costs, reaching an average annual cost of 550k€ for the setting with $J = 5$ and 1100k€ for the setting with $J = 10$. In contrast, DRLB incurs average costs of 450k€ and 800k€ for the same two settings, respectively. Furthermore, the distribution queue for both methods appears fatter in the second experiment, especially for PPO; this is a result of the increased complexity derived from a greater number of distribution warehouses for which determining an appropriate quantity to be shipped becomes more challenging, thus leading to higher costs.

The *computational times* for both training and testing are detailed in Table 5. As in the previous subsection, we report the training time (in minutes) needed to learn a policy and the testing time (in milliseconds) required to execute a single episode. In comparison to small settings, the training time increases by a factor of three to five. Notably, sQ faces the most significant increase, requiring a training time of 12.17 min for

the large setting experiment with $J = 10$, in contrast to approximately 2.40 min in the smaller setting experiments.

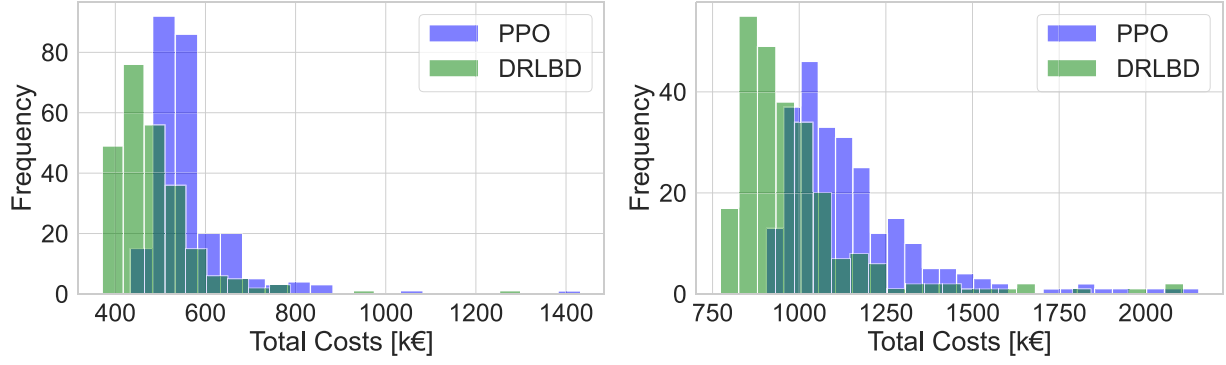
Despite the challenges posed by the large settings, all testing times remain under one second, thus proving that the primary computational bottleneck is the training phase of the algorithms. Moreover, such small computational times needed to compute actions allow for the execution of multiple experiments in a reasonable amount of time. This promptness is especially desirable for real-world applications, as decision-makers can swiftly conduct *what-if analyses*.

Among all methods, EVPI is most affected by the increment of J , increasing its computational time by 4.16 times. Finally, it is crucial to note that the testing times for DRLBD appear small; this can be attributed to the small number of stages and the variable relaxation applied to the mathematical model used to determine the shipping quantity, underscoring the DRLBD effectiveness also in this context.

5.3. Sensitivity analysis

To further investigate how demand variations affect the results described and commented in the previous subsection, we design and execute additional experiments, replicating those from Section 5.2 while using different parameter values. Specifically, we focus on the setting consisting of 5 distribution warehouses (i.e., $J = 5$), as it facilitates to perform numerical experiments that are both quick and easily manageable. The demand defined by Eq. (14) depends on several parameters, such as amplitude D , period P , and phase ϕ_j of the reference demand value, along with parameters r and p associated with the random noise ϵ . Since conducting a comprehensive analysis of the results involving all these parameters would be complex, we opted to provide only some select and possibly valuable insights here. Nevertheless, the publicly available software code allows replicating the results and considering different parameter values.

In detail, we introduced variations in P and D values, i.e., $P = \{3, 6, 9\}$ and $D = \{1, 2, 3\}$. By adopting these parameters, we can adjust the number of demand peaks throughout the time horizon ($T = 12$) as well as the amplitude of the demand. Additionally, varying D allows us to regulate the balance between deterministic and stochastic demand components. Compatible with the previous subsection, we keep $\phi_j = 0$, expressing the harshest condition, while we expand the capacities of the distribution warehouses (i.e., $I_j^{\max} = 8$) according to the maximum D value employed (i.e., $D = 3$). This choice guarantees a standardized setting across all experiments, preventing the risk of backlogs caused by insufficient storage restriction. All other values are consistent with those presented in Section 5 (and are reported in Appendix A). As in our



(a) Total costs histogram with $J = 5$ distribution warehouses. (b) Total costs histogram with $J = 10$ distribution warehouses.

Fig. 8. Total costs histograms over 250 episodes in relation to PPO and DRLBD techniques.

Table 5

Average training and testing times for the two experiments conducted under large settings (i.e., with $J = 5$ and $J = 10$ distribution warehouses, respectively) with their standard deviation (in brackets).

Algorithm	Large setting (with $J = 5$)		Large setting (with $J = 10$)	
	Training time [min]	Testing time [ms]	Training time [min]	Testing time [ms]
DRLBD	6.40 (0.01)	28 (1.96)	11.30 (0.01)	40 (2.14)
PPO	6.40 (0.01)	18 (1.23)	11.30 (0.01)	25 (1.56)
sQ	6.04 (0.01)	11 (0.51)	12.17 (0.01)	14 (0.77)
EVP	–	49 (0.58)	–	204 (0.32)
PI	–	245 (612)	–	362 (1627)

Table 6

Average EVPI gap (as a percentage) by varying demand parameters $P = \{3, 6, 9\}$ and $D = \{1, 2, 3\}$, and considering $J = 5$ distribution warehouses. Adjusting these parameters allows us to alter the number of demand peaks and the balance between deterministic and stochastic demand components. The standard deviation is provided in round brackets. The lower the value, the better the algorithm.

		$D = 1$	$D = 2$	$D = 3$
$P = 3$	DRLBD	155.98 (21)	112.23 (25)	30.85 (28)
	PPO	199.63 (31)	167.63 (47)	45.88 (38)
	sQ	234.11 (42)	197.91 (55)	70.68 (47)
	EVP	860.06 (290)	595.91 (170)	116.68 (72)
$P = 6$	DRLBD	151.77 (21)	105.01 (22)	31.78 (27)
	PPO	196.01 (34)	156.54 (42)	47.10 (36)
	sQ	228.05 (44)	223.23 (70)	75.49 (52)
	EVP	858.57 (288)	571.42 (168)	122.25 (71)
$P = 9$	DRLBD	164.53 (23)	120.70 (25)	51.22 (29)
	PPO	221.37 (39)	187.65 (56)	59.00 (33)
	sQ	244.44 (42)	195.58 (42)	96.40 (48)
	EVP	901.54 (300)	618.92 (184)	183.21 (71)

prior evaluations, we considered DRLBD, PPO, sQ, and EVP. In Table 6, we reported the relative gaps compared to PI (i.e., the EVPI gap).

The most challenging scenario for all techniques emerged when $P = 9$ and $D = 1$. In this configuration, a single demand peak appears over the time horizon, and the deterministic component is relatively small; this makes preventing backlogs particularly complex for all techniques. However, as the deterministic component grows, or as P decreases, the problem becomes more tractable since all methods begin to reduce costs. This outcome is reasonable as a greater deterministic component allows more accurate forecasting of the future. Moreover, when peaks occur more frequently, they can be captured with a shorter look-ahead interval, facilitating more effective production and shipping decisions. The most favorable scenario emerges when $D = 3$ and $P = 3$. Here, a demand peak appears every 3 time steps, and the demand is primarily

influenced by its deterministic component, thus promoting enhanced production and shipment planning.

It is interesting to notice that D affects the performance of the methods more than P . In fact, while the variation along the columns is around 30%, the variation along the rows is approximately 6%. The most substantial growth occurs when transitioning from $D = 2$ to $D = 3$; this is motivated by an increment of the deterministic demand component that enables all methods to make more knowledgeable decisions.

When comparing various techniques, it becomes evident that the proposed heuristic consistently outperforms its competitors in all settings. Among the others, PPO demonstrates superior results when compared to sQ and EVP, with sQ surpassing EVP. Interestingly, the most significant performance improvements across different D values are noticed with EVP, underscoring its heightened sensitivity to uncertainty. In contrast, the performance variations of the other methods do not reveal any substantial differences, thus confirming the conclusion that addressing uncertainty with appropriate methods is the proper approach.

6. Conclusions

In this paper, we introduce a novel heuristic for addressing the supply chain inventory management problem within a divergent two-echelon supply chain. The proposed heuristic *decomposes* the problem using a deep reinforcement learning algorithm to determine the number of batches to produce and multi-stage stochastic programming to establish the quantities of batches to ship to each distribution warehouse. In practical terms, this approach *combines* the strengths of both techniques: the model-based approach of MS programming for immediate logistics decisions and the simulation–optimization abilities of DRL to make production decisions that require a longer look ahead interval.

We assessed the performance of the proposed heuristic through a series of numerical experiments. In smaller settings, where computing an optimal solution was feasible, DRLBD demonstrated performance reasonably close to exact methods. Moreover, in larger and more complex settings where determining an optimal solution was impracticable, DRLBD performed robustly, consistently outperforming both the PPO algorithm and the (s, Q) -policy used as benchmarks. To further substantiate these findings, we conducted a sensitivity analysis by varying specific demand parameters. This analysis confirmed the stability of our results through different value combinations, also proving that experiments featuring a low frequency of peaks and a small deterministic component are more challenging to handle.

From a computational time point of view, DRLBD rapidly computes actions, with the DRL training phase emerging as the primary potential bottleneck. This efficiency enables decision-makers to employ the proposed heuristic as an effective tool to conduct what-if analysis in a reasonable amount of time.

In conclusion, these findings underscore the potential of the proposed heuristic in adeptly addressing the SCIM problem across a range of distinct scenarios. Future research directions will explore more complex supply chain environments, encompassing critical factors like fixed production costs, uncertainty in item availability for the factory, and other relevant aspects. Lastly, investigations will extend to supply chain settings with more than two echelons to better evaluate the applicability and performance of the proposed heuristic. Through these efforts, we aspire to advance the understanding of the SCIM problem in increasingly complex and dynamic environments.

Declaration of competing interest

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

Data availability

The datasets generated and analyzed during the current study can be generated using the open-source software library developed for this research, which can be accessed through GitHub (<https://github.com/frenkowski/SCIMAI-Gym>).

Acknowledgments

We are grateful to the reviewers, whose insightful comments have significantly contributed to the enhancement of this paper. Furthermore, we extend our thanks to Enrico Robbiano (Senior Manager - Supply Chain Analytics at Bristol-Myers Squibb) for his contribution to validating the data and Giovanni Fantoni for his invaluable support in improving the source code.

Funding

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Appendix A. SCIM environment parameters

Table 7 outlines the parameters defining the SCIM environment for small and large settings. For small experiments, the indication of two values is associated with $\epsilon \sim B(0.5)$ and ϵ as in Eq. (16), respectively. For large experiments, the exhibition of two values corresponds to $J = 5$ and $J = 10$ distribution warehouses, respectively. This comparative methodology enables the evaluation of the impact of distinct configurations on the overall performance and efficiency of our proposed techniques.

Table 7

Parameters defining the SCIM environment for small and large settings. In small settings, when two values are listed, the first is associated with the experiment with $\epsilon \sim B(0.5)$, while the second refers to ϵ as in Eq. (16). For the large settings, the provided two values correspond to experiments with $J = 5$ and $J = 10$ distribution warehouses, respectively.

Parameters	Small settings	Large settings
Maximum demand value	5	2
Production costs	1	1
Vehicles capacities	3	2
Logistic costs fixed	0.7	random(0.7, 1)
Logistic costs variable	0.03	random(0.01, 0.07)
Maximum production	{8, 15}	{13, 25}
Factory capacities	{10, 20}	{18, 36}
Distribution warehouses capacities	{5, 10}	{6, 6}
Factory storage costs	0.1	random(0.01, 0.1)
Distribution warehouses storage costs	1	random(1, 2)
Backorder costs	10	10

More specifically, in small settings, the demand amplitude remains constant at 5, whereas in large ones, it is set at 2. Production costs are consistent across experiments, with a value of 1. Vehicle capacities differ between the two settings. Small experiments accommodate a capacity of 3, with a fixed cost per vehicle of 0.7 and a variable cost per batch of 0.03. Conversely, large experiments allow for a capacity of 2, with fixed costs randomly ranging between 0.7 and 1 and variable costs that exhibit random variation within the range of 0.01 and 0.07.

Maximum production capacity also exhibits different values, with bounds set at 8 and 15 batches for small settings and 13 and 25 batches for large ones. Furthermore, factory capacities display differences, with values of 10 and 20 batches in small settings and 18 and 36 batches in large ones. Distribution warehouse capacities are lower than factory capacities, with 5 and 10 batches for small settings and large ones that consistently maintain a capacity of 6 batches. It is crucial to highlight that these configurations are strategic and designed to encourage agents to preserve stocks in advance and effectively avoid myopic behavior.

Concerning storage costs, factory storage costs are set to 0.1 in small experiments, while they randomly vary between 0.01 and 0.1 in large ones. In contrast, distribution warehouse storage costs remain constant at 1 for small settings and exhibit random variations between 1 and 2 for large settings. Finally, backorder costs remain fixed at 10 across all experiments.

Appendix B. Hyperparameters selection

Selecting appropriate values for hyperparameters in neural networks is complex and time-consuming, as extensively discussed in the relevant literature (Feurer and Hutter, 2019; Yang and Shami, 2020). Hyperparameters play a crucial role in the context of DRL algorithms since they can significantly influence training and, consequently, relative performance (Boute et al., 2022). Our choice of hyperparameters is based on the Ray documentation and in observance of the discussions presented in Gijbrecchts et al. (2022) and Stranieri and Stella (2022).

Table 8 lists the selected hyperparameters for the underlying multi-layer perceptron with two hidden layers, along with their corresponding values. Through a grid search, the PPO algorithm was trained for 75k episodes for the two experiments belonging to the small settings, 100k episodes for the experiment of the large setting comprising 5 distribution warehouses (i.e., $J = 5$), and 200k episodes for the experiment of the large setting involving 10 distribution warehouses (i.e., $J = 10$). The results presented in the paper refer to the best combination of hyperparameters identified, for which we replicated the training and testing procedures.

Moreover, we assess the PPO agent to access the demand values of the previous timesteps (i.e., referring to Eq. (3), we set $\tau = 2$ in small experiments and $\tau = 3$ in large ones). This choice reflects a balance between providing the PPO agent with enough information to make

Table 8

Hyperparameters of the PPO algorithm selected for tuning, along with their corresponding values. Through a grid search, we search for the best combination of hyperparameters for each experiment.

PPO hyperparameters	Values
Neurons per hidden layer	{(16, 16), (32, 32), (64, 64), (128, 128)}
Learning rate	{5e-4, 5e-5, 5e-6}
Train batch size	{200, 400, 800, 2000, 4000, 8000}
Stochastic gradient descent mini-batch size	{64, 128, 256, 512}
Stochastic gradient descent iterations	{15, 30, 45}

informed decision-making while not overwhelming it with excessive historical data, which might introduce unnecessary complexity and longer training times.

The parameters governing the (s, Q) -policy are determined using a data-driven approach based on Bayesian optimization, as presented in Stranieri and Stella (2022).

Finally, DRLBD exploits the best combination of PPO hyperparameters as previously described, while to generate the scenario tree it employs a Monte Carlo technique known as moment matching.

References

- Agrawal, N., Smith, S.A., 1996. Estimating negative binomial demand for retail inventory management with unobservable lost sales. *Nav. Res. Logist.* 43 (6), 839–861. [http://dx.doi.org/10.1002/\(sici\)1520-6750\(199609\)43:6<839::aid-nav4>3.0.co;2-5](http://dx.doi.org/10.1002/(sici)1520-6750(199609)43:6<839::aid-nav4>3.0.co;2-5).
- Alonso-Ayuso, A., Escudero, L., Garín, A., Ortuño, M., Pérez, G., 2003. An approach for strategic supply chain planning under uncertainty based on stochastic 0-1 programming. *J. Global Optim.* 26 (1), 97–124. <http://dx.doi.org/10.1023/a:1023071216923>.
- Aviv, Y., Federgruen, A., 1997. Stochastic inventory models with limited production capacity and periodically varying parameters. *Probab. Engng. Inform. Sci.* 11 (1), 107–135. <http://dx.doi.org/10.1017/s026996480000471x>.
- Bertsekas, D., 2021. *Rollout, Policy Iteration, and Distributed Reinforcement Learning*. Athena Scientific.
- Birge, J.R., Louveaux, F., 2011. *Introduction to Stochastic Programming*. Springer New York, <http://dx.doi.org/10.1007/978-1-4614-0237-4>.
- Boute, R.N., Gijbrecchts, J., van Jaarsveld, W., Vanvuchelen, N., 2022. Deep reinforcement learning for inventory control: A roadmap. *European J. Oper. Res.* 298 (2), 401–412. <http://dx.doi.org/10.1016/j.ejor.2021.07.016>.
- Brandimarte, P., 2006. Multi-item capacitated lot-sizing with demand uncertainty. *Int. J. Prod. Res.* 44 (15), 2997–3022. <http://dx.doi.org/10.1080/00207540500435116>.
- Brandimarte, P., 2011. *Quantitative Methods*. Wiley, <http://dx.doi.org/10.1002/9781118023525>.
- Brandimarte, P., 2014. *Handbook in Monte Carlo Simulation: Applications in Financial Engineering, Risk Management, and Economics*. John Wiley & Sons.
- Brandimarte, P., 2021. *From Shortest Paths to Reinforcement Learning*. Springer International Publishing, <http://dx.doi.org/10.1007/978-3-030-61867-4>.
- Brandimarte, P., Zotteri, G., 2007. *Introduction to Distribution Logistics*. Wiley, <http://dx.doi.org/10.1002/9780470170052>.
- Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W., 2016. OpenAI gym. <http://dx.doi.org/10.48550/ARXIV.1606.01540>, URL: <https://arxiv.org/abs/1606.01540>.
- Chaharsooghi, S.K., Heydari, J., Zegordi, S.H., 2008. A reinforcement learning model for supply chain ordering management: An application to the beer game. *Decis. Support Syst.* 45 (4), 949–959. <http://dx.doi.org/10.1016/j.dss.2008.03.007>.
- Dulac-Arnold, G., Levine, N., Mankowitz, D.J., Li, J., Paduraru, C., Goyal, S., Hester, T., 2021. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Mach. Learn.* 110 (9), 2419–2468. <http://dx.doi.org/10.1007/s10994-021-05961-4>.
- Feurer, M., Hutter, F., 2019. *Hyperparameter optimization*. In: *Automated Machine Learning*. Springer International Publishing, pp. 3–33. http://dx.doi.org/10.1007/978-3-030-05318-5_1.
- Fujimoto, S., van Hoof, H., Meger, D., 2018. Addressing function approximation error in actor-critic methods. In: Dy, J., Krause, A. (Eds.), *Proceedings of the 35th International Conference on Machine Learning*. In: *Proceedings of Machine Learning Research*, vol. 80, PMLR, pp. 1587–1596, URL: <https://proceedings.mlr.press/v80/fujimoto18a.html>.
- Gijbrecchts, J., Boute, R.N., Mieghem, J.A.V., Zhang, D.J., 2022. Can deep reinforcement learning improve inventory management? performance on lost sales, dual-sourcing, and multi-echelon problems. *Manuf. Serv. Oper. Manag.* 24 (3), 1349–1368. <http://dx.doi.org/10.1287/msom.2021.1064>.
- Grewal, C.S., Enns, S., Rogers, P., 2015. Dynamic reorder point replenishment strategies for a capacitated supply chain with seasonal demand. *Comput. Ind. Eng.* 80, 97–110. <http://dx.doi.org/10.1016/j.cie.2014.11.009>.
- Gurobi Optimization, LLC, 2023. Gurobi optimizer reference manual. URL: <https://www.gurobi.com>.
- Harsha, P., Jagmohan, A., Kalagnanam, J., Quanz, B., Singhvi, D., 2021. Math programming based reinforcement learning for multi-echelon inventory management. *SSRN Electron. J.* <http://dx.doi.org/10.2139/ssrn.3901070>.
- Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., Meger, D., 2018. *Deep reinforcement learning that matters*. In: AAAI'18/IAAI'18/EAAI'18, AAAI Press.
- Huang, Y., Chen, C.-W., Fan, Y., 2010. Multistage optimization of the supply chains of biofuels. *Transp. Res. E* 46 (6), 820–830. <http://dx.doi.org/10.1016/j.tre.2010.03.002>.
- Hubbs, C.D., Perez, H.D., Sarwar, O., Sahinidis, N.V., Grossmann, I.E., Wassick, J.M., 2020. OR-gym: A reinforcement learning library for operations research problems. <http://dx.doi.org/10.48550/ARXIV.2008.06319>, URL: <https://arxiv.org/abs/2008.06319>.
- Kemmer, L., von Kleist, H., de Rochebouët, D., Tziortziotis, N., Read, J., 2018. *Reinforcement learning for supply chain optimization*. In: *European Workshop on Reinforcement Learning*, Vol. 14.
- Khouja, M., 2003. Optimizing inventory decisions in a multi-stage multi-customer supply chain. *Transp. Res. E* 39 (3), 193–208. [http://dx.doi.org/10.1016/s1366-5545\(02\)00036-4](http://dx.doi.org/10.1016/s1366-5545(02)00036-4).
- de Kok, T., Grob, C., Laumanns, M., Minner, S., Rambau, J., Schade, K., 2018. A typology and literature review on stochastic multi-echelon inventory models. *European J. Oper. Res.* 269 (3), 955–983. <http://dx.doi.org/10.1016/j.ejor.2018.02.047>.
- Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z., Paul, W., Jordan, M.I., Stoica, I., 2017. Ray: A distributed framework for emerging AI applications. <http://dx.doi.org/10.48550/ARXIV.1712.05889>, URL: <https://arxiv.org/abs/1712.05889>.
- Mortazavi, A., Khamseh, A.A., Azimi, P., 2015. Designing of an intelligent self-adaptive model for supply chain ordering management system. *Eng. Appl. Artif. Intell.* 37, 207–220. <http://dx.doi.org/10.1016/j.engappai.2014.09.004>.
- 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). IEEE, <http://dx.doi.org/10.1109/cac48633.2019.8997498>.
- Powell, W.B., 2011. *Approximate Dynamic Programming*. John Wiley & Sons, Inc., <http://dx.doi.org/10.1002/9781118029176>.
- Preusser, M., Almeder, C., Hartl, R.F., Klug, M., 2010. LP modelling and simulation of supply chain networks. In: *Supply Chain Management und Logistik*. Physica-Verlag, pp. 95–113. http://dx.doi.org/10.1007/3-7908-1625-6_4.
- Ravulapati, K.K., Rao, J., Das, T.K., 2004. A reinforcement learning approach to stochastic business games. *IIE Trans.* 36 (4), 373–385. <http://dx.doi.org/10.1080/07408170490278698>.
- Rolf, B., Jackson, I., Müller, M., Lang, S., Reggelen, T., Ivanov, D., 2022. A review on reinforcement learning algorithms and applications in supply chain management. *Int. J. Prod. Res.* 61 (20), 7151–7179. <http://dx.doi.org/10.1080/00207543.2022.2140221>.
- Roy, B.V., Bertsekas, D., Lee, Y., Tsitsiklis, J., 1997. A neuro-dynamic programming approach to retailer inventory management. In: *Proceedings of the 36th IEEE Conference on Decision and Control*. IEEE, <http://dx.doi.org/10.1109/cdc.1997.652501>.
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms. <http://dx.doi.org/10.48550/ARXIV.1707.06347>, URL: <https://arxiv.org/abs/1707.06347>.
- Stranieri, F., Stella, F., 2022. Comparing deep reinforcement learning algorithms in two-echelon supply chains. <http://dx.doi.org/10.48550/ARXIV.2204.09603>, <https://arxiv.org/abs/2204.09603>.
- Stranieri, F., Stella, F., Kouki, C., 2023. Performance of deep reinforcement learning algorithms in two-echelon inventory control systems. *Int. J. Prod. Res.* Submitted for Publication.

- Sui, Z., Gosavi, A., Lin, L., 2010. A reinforcement learning approach for inventory replenishment in vendor-managed inventory systems with consignment inventory. *Eng. Manage. J.* 22 (4), 44–53. <http://dx.doi.org/10.1080/10429247.2010.11431878>.
- Sutton, R.S., Barto, A.G., 2018. *Reinforcement Learning: An Introduction*. MIT Press.
- Vanvuchelen, N., Boute, R.N., 2022. The use of continuous action representations to scale deep reinforcement learning: An application to inventory control. *SSRN Electron. J.* <http://dx.doi.org/10.2139/ssrn.4253600>.
- Vincent, P., 1985. Exact fill rates for items with erratic demand patterns. *INFOR: Inf. Syst. Oper. Res.* 23 (2), 171–181. <http://dx.doi.org/10.1080/03155986.1985.11731953>.
- Wagner, H.M., Whitin, T.M., 1958. Dynamic version of the economic lot size model. *Manage. Sci.* 5 (1), 89–96. <http://dx.doi.org/10.1287/mnsc.5.1.89>.
- Yan, Y., Chow, A.H., Ho, C.P., Kuo, Y.-H., Wu, Q., Ying, C., 2022. Reinforcement learning for logistics and supply chain management: Methodologies, state of the art, and future opportunities. *Transp. Res. E* 162, 102712. <http://dx.doi.org/10.1016/j.tre.2022.102712>.
- Yang, L., Shami, A., 2020. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing* 415, 295–316. <http://dx.doi.org/10.1016/j.neucom.2020.07.061>.
- Zipkin, P., 2000. *Foundations of Inventory Management*. Irwin Professional Publishing, Maidenhead, England.