Comparing Deep Reinforcement Learning Algorithms in Two-Echelon Supply Chains

Comparing Deep Reinforcement Learning Algorithms in Two-Echelon Supply Chains

Francesco Stranieri*[1,2][0000−0002−5366−8499] and
Fabio Stella[1][0000−0002−1394−0507]

[1] University of Milan-Bicocca, Milan MI 20125, Italy
[2] Polytechnic of Turin, Turin TO 10129, Italy

Abstract. In this study, we analyze and compare the performance of state-of-the-art deep reinforcement learning algorithms for solving the supply chain inventory management problem. This complex sequential decision-making problem consists of determining the optimal quantity of products to be produced and shipped across different warehouses over a given time horizon. In particular, we present a mathematical formulation of a two-echelon supply chain environment with stochastic and seasonal demand, which allows managing an arbitrary number of warehouses and product types. Through a rich set of numerical experiments, we compare the performance of different deep reinforcement learning algorithms under various supply chain structures, topologies, demands, capacities, and costs. The results of the experimental plan indicate that deep reinforcement learning algorithms outperform traditional inventory management strategies, such as the static (s, Q)-policy. Furthermore, this study provides detailed insight into the design and development of an open-source software library that provides a customizable environment for solving the supply chain inventory management problem using a wide range of data-driven approaches.

Keywords: artificial intelligence · deep learning · reinforcement learning · smart manufacturing · inventory management.

1 Introduction

Supply chain inventory management (SCIM) is a *sequential decision-making problem* consisting of determining the optimal quantity of products to produce at the factory and to ship to different distribution warehouses over a given time horizon. As evidenced by the helpful roadmap of [2], deep reinforcement learning (DRL) algorithms are rarely applied to the SCIM field, although they can be used to develop near-optimal policies that are difficult, or impossible at worst, to achieve using traditional methods. Indeed, the uncertain and stochastic nature of products demand, as well as lead times, represent significant obstacles for mathematical programming approaches to be effective, with specific reference

* Corresponding author. E-mail: francesco.stranieri@polito.it.

to those cases where the modeling of SCIM's entities is reasonable, for example, assuming a finite capacity of warehouses [9].

Regarding the DRL algorithms that have been currently applied to tackle the SCIM problem, we found that they suffer the following *limitations*: i) given a supply chain *structure* (e.g., divergent [3] two-echelon [4]), no DRL algorithm has been deeply tested with respect to different *topologies* (i.e., by changing the number of warehouses); ii) no extensive experiments have been performed on the same supply chain structure by varying different *configurations* (e.g., demands, capacities, and costs); iii) no extension has been proposed for *comparing different DRL algorithms* and determining which one is more appropriate for a particular supply chain topology and configuration, as suggested by [1,2].

Furthermore, relevant aspects of the SCIM problem have not yet been addressed efficiently [23], for example: i) the *sequence of events* required to reproduce and validate a simulation model is not always well-defined or given. Hence, making available a consistent and universal open-source SCIM environment can improve reusability and reproducibility, especially if implemented with standard APIs (like those of OpenAI Gym [5]). In this way, it is also possible to import DRL algorithms from reliable libraries and focus solely on their fine-tuning, instead of developing them from scratch; ii) DRL algorithms are typically compared with some standard *static reorder policies*. However, their performances are not always compared with those achieved by an oracle, i.e., a baseline who knows the optimal action to take a priori, thus making it difficult to evaluate the DRL effectiveness in real-world environments (the only paper in which an oracle is introduced is [6]); iii) none of the DRL papers available in the specialized literature considers a *multi-product approach*, whereas it has been considered relating to other solution methods [23]. Considering more than one product type increases the dimensionality and complexity of the problem, consequently requiring an efficient implementation of the SCIM environment and DRL algorithms.

This paper makes the following *contributions* to the SCIM decision-making problem:

- Design and formulation of a stochastic and divergent two-echelon SCIM environment under seasonal demand, which allows an arbitrary number of warehouses and product types to be managed.
- Comparison of a set of state-of-the-art DRL algorithms in terms of their ability to find an optimal policy, i.e., a policy which maximizes the SCIM's profit as achieved by an oracle.

[3] In a *linear* supply chain, each participant has one predecessor and one successor; in a *divergent* supply chain, each has one predecessor but can have multiple successors, while the opposite is true in a *convergent* supply chain. Finally, in a *general* supply chain, each participant can have several predecessors and several successors.

[4] A supply chain can include multiple stages, called formally *echelons*, through which the stocks are moved to reach the customer. When the number of echelons is greater than one, we refer to a *multi-echelon* supply chain.

- Evaluation of performances achieved by state-of-the-art DRL algorithms and comparison to a static reorder policy, i.e., an (s, Q)-policy, whose optimal parameters have been set through a data-driven approach.
- Design and run of a rich experimental plan involving different SCIM topologies and configurations as well as values of hyperparameters associated with DRL algorithms'.
- Design and development of an open-source library for solving the SCIM problem [6], thus embracing the open science principles and guaranteeing reproducible results.

The rest of the paper is organized as follows: Section 2 is devoted to introducing and providing main reinforcement learning (RL) definitions and notation, also highlighting how RL approaches have dealt with the SCIM problem; in this section, we also describe the state-of-the-art DRL algorithms and how they have been used to address the SCIM problem. Section 3 describes the main methodological contributions of this paper. The rich experimental plan is then reported in Section 4, while the results of numerical experiments are presented in Section 5. Lastly, discussions and conclusions are given in Section 6.

2 Literature Review

While reinforcement learning has recently achieved remarkable results in the field of artificial intelligence, mainly when applied to video games and gaming in a more general sense [13,18,20], its deployment in industrial settings has been less extensive. Despite RL proving to be effective in solving complex sequential decision-making problems, its translation into industrial use cases is still emerging, devising a concrete opportunity for further explore its potentialities [23].

Essentially, RL adopts the Markov Decision Process (MDP) framework to represent the interactions between a learning agent and an environment [19]. As shown in Figure 1, at each time step t, the agent observes the current state of the environment, $S_t \in \mathcal{S}$, chooses an action, $A_t \in \mathcal{A}(S_t)$, and obtains a reward, $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$; then, the environment transitions into a new state, S_{t+1}. The goal of RL is thus to find an optimal policy, $\pi_* : \mathcal{S} \to \mathcal{A}$, that maximizes the *expected discounted return*, $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$, where $0 \leq \gamma \leq 1$ is a hyperparameter called *discount rate*.

One of the most common approaches for solving the SCIM problem through RL algorithms turns out to be Q-learning. This approach is based on a tabular and temporal-difference (TD) algorithm that learns how to determine the *value* of an action A_t in a state S_t, referred to as the Q-value, in accordance with the following update rule: $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\delta_t$, where $0 \leq \alpha \leq 1$ is a hyperparameter called *learning rate*, and $\delta_t = [R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$ is the TD error. Q-values of each state-action pair are stored in a table, known as Q-table, where each state is represented by a row and each action by a

[5] The OpenAI Gym library is available on https://www.gymlibrary.dev.
[6] Our open-source library is available on https://github.com/frenkowski/SCIMAI-Gym.

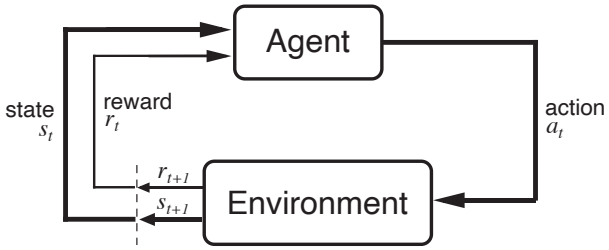

Fig. 1. Agent-environment interface in an MDP (taken from [19]).

column. Through the Q-learning algorithm, Q-values associated with each state-action pair are estimated and, once convergence has been achieved (which is guaranteed under certain conditions [7]), an optimal policy can be easily obtained by identifying, for each state, the action with the highest Q-value, that is, $\pi_*(S_t) = \arg\max_a Q(S_t, a)$.

In [3], which is one of the most cited RL articles about SCIM, the authors proposed an approach based on Q-learning to address (a *centralized* variant of) the SCIM problem consisting of a linear supply chain with four participants. In particular, they defined the current system state as a vector consisting of the four inventory positions in terms of current stock levels. However, considering that inventory positions thus defined may take infinite values, applying this strategy appears unfeasible since the Q-table would be in turn infinite. Consequently, the authors *discretized* the state space into nine intervals. In this way, the possible state values amount to 9^4. Regarding actions, their approach determines the number of products to order via the d+x policy; precisely, if a participant in the previous time step received a request for d product units from the succeeding stage, the d+x *policy* requires ordering d+x units to the preceding stage in the current time step. The learning process's objective is hence to determine the value of the unknown variable x according to the given system state. For limiting the Q-table size, x was *constrained* by the authors to belong to [0, 3] so that the possible number of actions amounts to 4^4.

Obviously, by defining restricted state and action spaces, the resulting Q-table appears to be more manageable. However, analyzing various RL studies [23], it becomes evident that the Q-tables implemented are typically huge and, thus, *unscalable*. For example, the Q-table adopted by [3] has a number of cells equal to $(9^4 \cdot 4^4 =)$ 1 679 616, equivalent to the number of states multiplied by the number of actions. Consequently, expanding the size of the state or action spaces might not be feasible, as the Q-tables can no longer be handled.

Consequently, tabular RL methods can only be applied to discretized or constrained state and action spaces. However, discretization leads to a *loss of crucial information*, in addition to being unsuitable for real-world scenarios; thus, we need improved RL methods to address the SCIM problem effectively.

In this respect, deep reinforcement learning is a combination of RL with deep learning (DL) which promises to scale to previously intractable decision-making

problems, i.e., environments with high dimensional state and action spaces. DL is rooted into artificial neural networks (ANNs) [10], which are universal approximators capable of providing an optimal approximation of *highly nonlinear functions*. In practice, function parameters θ are adjusted during the learning process in order to maximize the expected return (or, alternatively, to minimize the TD error).

The DRL algorithms we implemented belong to the policy-based methods, which can learn a *parameterized* and stochastic policy, $\pi_\theta \approx \pi_*$ with $\pi : \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$, to select actions directly (as opposed to the Q-learning algorithm, which is part of value-based methods [19]). Inside them, policy gradient methods offer a considerable theoretical advantage through the *policy gradient theorem*, and the vanilla policy gradient (VPG) algorithm [21] is a natural result of this theorem; however, the *high variance* of gradient estimates usually results in policy update instabilities [22]. Also to mitigate this issue, [16] proposed an actor-critic algorithm (which means that a policy and a value function are simultaneously learned) called trust region policy optimization (TRPO), which bounds the difference between the new and the old policy in a *trust region*. Proximal policy optimization (PPO) [17] shares the same background as TRPO, but has demonstrated comparable or superior performance while being significantly *simpler* to implement and tune. Asynchronous advantage actor-critic (A3C) [12] is also one of the available state-of-the-art actor-critic algorithms. Its core idea is to have different agents interacting with different representations of the environment, each with its parameters. Periodically (and asynchronously), they update a global ANN that incorporates *shared parameters*. For interested readers, an in-depth and more rigorous discussion on the various DRL algorithms can be found in [4].

To the best of the authors' knowledge, only few papers have implemented DRL algorithms to solve the SCIM problem, despite some restrictions. More in detail, an *extension* of deep Q-network (DQN) [13] has been proposed in [14] to solve (a *decentralized* variant of) the SCIM problem. The authors revealed that a DQN agent, which basically involves an ANN instead of a Q-table to return the Q-value for a state-action pair, can learn a near-optimal policy when other supply chain participants follow a base-stock policy; under a *base-stock policy*, each participant orders in each time step t a quantity to bring its stocks equal to a fixed number s, known as the base-stock level, to determine in an optimal way. Because DQN requires a restricted action space cardinality, the authors performed numerical experiments using a $d+x$ policy, with x constrained to one of the following intervals: [-2, +2], [-5, +5], and [-8, +8].

Alternatively, authors in [15] proposed the VPG algorithm to address a two-echelon supply chain with stochastic and seasonal demand. Due to storage capacity constraints, the authors designed a *dynamic action space*. As a result, the number of products to ship is determined also by considering the number of stocks actually present in the warehouses. To evaluate the VPG performance, three different numerical experiments are presented, and the results show that the VPG agent is able to outperform the (s, Q)-policy employed as a baseline in all three experiments. In this context, the *(s, Q)-policy* can be expressed by a

rule: at each time step t, the current stock level is compared to the reorder point s. If the stock level falls below the reorder point s, then the (s, Q)-policy orders Q units of product; otherwise, it does not take any action. Also in this case, the parameters s and Q are to be determined optimally.

Using the same supply chain structure but with ten warehouses and a normal distribution, authors in [5] applied and tuned the A3C algorithm for two different numerical experiments. The authors restricted the action space by implementing a *state-dependent* base-stock policy, and the results show that A3C can achieve performance comparable to state-of-the-art heuristics and approximate dynamic programming algorithms, despite its initial tuning remaining computationally intensive.

Finally, in the experimental scenario analyzed by [1], a general four-echelon supply chain with two nodes per echelon is presented. The system state consists of product quantity currently available and in transit across the supply chain, plus *future* customer demands. To deal with the optimization problem, the authors proposed the PPO algorithm, while a deterministic linear programming agent (i.e., considering a deterministic demand) is employed as a baseline. Results of numerical experiments show that PPO still achieves satisfactory results.

3 Problem Definition

The SCIM environment we propose is primarily motivated by what was presented and discussed in [8,15]. Inspired by these works, we designed a divergent two-echelon supply chain that includes a *factory* that can produce various *product types*, a *factory warehouse*, and a certain number of *distribution warehouses*; an example of this structure is shown in Figure 2.

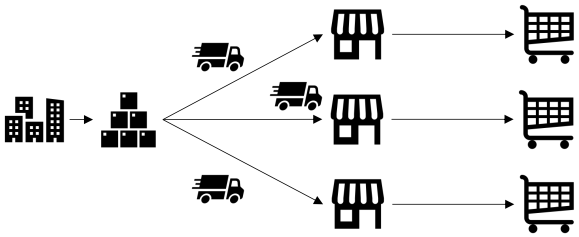

Fig. 2. A divergent two-echelon supply chain consisting of a factory and its warehouse (first echelon), plus three distribution warehouses (second echelon). Shopping carts represent customers' demands.

In our formulation, we assume that the factory produces I different product types. For each product type i, the factory decides, at every time step t, its respective production level $a_{i,0,t}$ (we assume $j = 0$ for the factory and $1 \leq j \leq J$

for the distribution warehouses), that is, how many units to produce, considering a fixed production cost of $z_{i,0}$ per unit. Moreover, the factory warehouse is associated with a maximum capacity of $c_{i,0}$ units for each product type i (this means that the overall capacity is given by $\sum_{i=0}^{I} c_{i,0} = c_0$). The cost of storing one unit of product type i at the factory warehouse is $z_{i,0}^S$ per time step, while the corresponding stock level at time t equals $q_{i,0,t}$. At every time step t, $a_{i,j,t}$ units of product type i are shipped from the factory warehouse to the distribution warehouse j, with an associated transportation cost of $z_{i,j}^T$ per unit. For each product type i, each distribution warehouse j has a maximum capacity of $c_{i,j}$ ($\sum_{i=0}^{I} c_{i,j} = c_j$), a storage cost of $z_{i,j}^S$ per unit, and a stock level at time t equal to $q_{i,j,t}$. The demand for product type i at distribution warehouse j for time step t is equivalent to $d_{i,j,t}$ units, while each unit of product type i is sold to customers at sale price p_i (which is identical across all warehouses).

Products are non-perishable and provided in discrete quantities. Additionally, we assume that each warehouse is legally obligated to fulfill all the submitted orders. Consequently, if an order for a certain time step exceeds the corresponding stock level, a penalty cost per unsatisfied unit is applied (the penalty cost for product type i is obtained by multiplying the penalty coefficient z_i^P by the sale price value p_i). Unsatisfied orders are maintained over time, and we design them as a negative stock level (which corresponds to *backordering*); this also implies that when the penalty coefficient is particularly high (e.g., $z_i^P \geq 1$), the agent may not be able to generate a positive profit if it causes backlog orders. Consequently, it should prefer a policy that leads to accumulating stocks in advance in order to pay storage costs rather than penalty costs.

3.1 Environment Formulation

In this subsection, we formalize the RL problem as an MDP. More precisely, we introduce and define the *main components* of the SCIM environment that we propose in this paper: the state vector, the action vector, and the reward function.

The *state vector* includes all current stock levels for each warehouse and product type, plus the last τ demand values, and is defined as follows:

$$\mathrm{s}_t = (q_{0,0,t}, \ldots, q_{I,J,t}, d_{t-\tau}, \ldots, d_{t-1}) \,,$$

where $d_{t-1} = (d_{0,1,t-1}, \ldots, d_{I,J,t-1})$. It is worth noticing that the actual demand d_t for the current time step t will not be known until the next time step $t+1$. This implementation choice ensures that the agent may benefit from learning the demand pattern so as to integrate a sort of *demand forecasting* directly into the policy. Additionally, we include the last demand values in order to enable the agent to have *limited knowledge* about the demand history and, consequently, to gain a basic comprehension of its fluctuations (similar to what was made originally by [8]). In our SCIM implementation, the agent can access the demand values of the last five time steps, even if preliminary results suggest that comparable performances are obtained by accessing the last three or four time steps.

Regarding the *action vector*, we chose to implement a *continuous action space* (i.e., the ANN generates the action value directly) consisting, for each product type, of the number of units to produce at the factory and of the number of units to ship to each distribution warehouse:

$$\mathrm{a}_t = (a_{0,0,t}, \ldots, a_{I,J,t})\,. \tag{1}$$

Usually, a relatively small and *identical upper bound* is typically adopted for all the action values to reduce the computational effort. However, the drawback is that this might lead to a significant drop in terms of performance. Indeed, if the upper bound is set too small, the agent may select an inefficient action given that the optimal one is outside the admissible range. Otherwise, if the upper bound is set too high, the agent may repeatedly choose an incoherent action, i.e., one that falls within the admissible range but exceeds a specified maximum capacity, consequently slowing down the training process.

Our implementation thus provides a continuous action space with an *independent upper bound* for each action value, in order to find a trade-off between efficiency and performance. In practical terms, the lower bound for each value is simply zero. In fact, it would be illogical to produce or ship negative quantities of products. Conversely, the upper bound for each distribution warehouse corresponds to its maximum capacity with respect to each product type (by referring to Equation (1), $0 \leq a_{i,j,t} \leq c_{i,j}$). To guarantee that the factory can adequately handle the various demands, its upper bound amounts to the sum of all warehouses' capacities with regard to each product type ($0 \leq a_{i,0,t} \leq \sum_{j=0}^{J} c_{i,j}$). We expect to improve both efficiency and performance with this intuition, as the action space is bounded (and hence restricted) but contains only coherent (and possibly optimal) actions. We specify that available stocks are not explicitly considered when the agent chooses an action. However, producing or shipping a number of stocks that it is not possible to store leads to a cost and, therefore, an *implicit penalty* for the agent. We also assume that there are no lead times both for production and transportation (or, to refer to the literature, we consider *constant lead times equal to 0*). This assumption allows us to isolate the primary dynamics of the problem without the additional effects of lead times, thus making the problem easier to address and manage.

To evaluate the performance of the DRL agents, we simulate a seasonal behavior by representing the *demand* as a co-sinusoidal function with a stochastic component, defined according to the following equation:

$$d_{i,j,t} = \left\lfloor \frac{d_{max_i}}{2} \left(1 + \cos\left(\frac{4\pi(2ij + t)}{T}\right)\right) + \mathcal{U}\left(0, d_{var_i}\right) \right\rfloor, \tag{2}$$

where $\lfloor \cdot \rfloor$ is the floor function, d_{max_i} is the maximum demand value for each product type, $\mathcal{U}$ is a random variable uniformly distributed on the support $(0, d_{var_i})$ representing the demand variations (i.e., the *uncertainty*), and T is the final time step of the episode. At each time step t, the demand may vary for each distribution warehouse j and product type i while maintaining the same behavior, as can be seen in Figure 3.

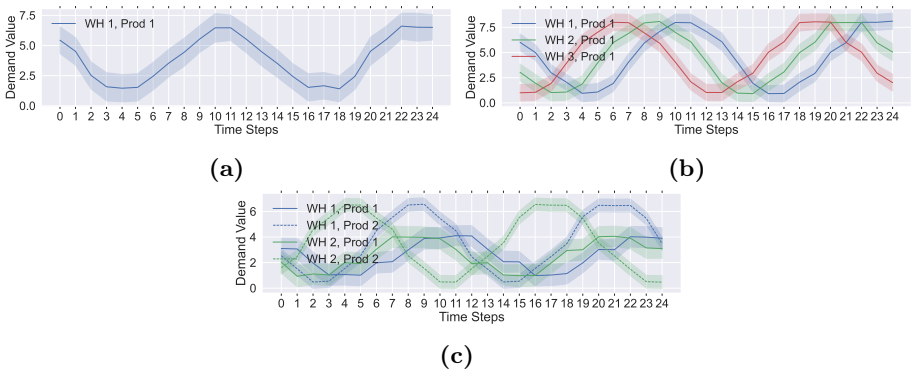

Fig. 3. Some instances of different demands behavior generated according to Equation (2) for different topologies and configurations of the SCIM problem: (a) one product type and one distribution warehouse with $d_{max} = 5$ and $d_{var} = 3$; (b) one product type and three distribution warehouses with $d_{max} = 7$ and $d_{var} = 2$; and (c) two product types and two distribution warehouses with $d_{max} = (3, 6)$ and $d_{var} = (2, 1)$ (referring to the values in round brackets, the first denotes the first product type, whereas the second indicates the second product type).

The *reward function* for each time step t is then defined as follows:

$$\begin{aligned} \mathrm{r}_t = &\sum_{j=1}^{J}\sum_{i=0}^{I} p_i \cdot d_{i,j,t} - \sum_{i=0}^{I} z_{i,0} \cdot a_{i,0,t} - \sum_{j=1}^{J}\sum_{i=0}^{I} z_{i,j}^{T} \cdot a_{i,j,t} \\ &- \sum_{j=0}^{J}\sum_{i=0}^{I} z_{i,j}^{S} \cdot \max(q_{i,j,t}, 0) + \sum_{j=0}^{J}\sum_{i=0}^{I} z_{i}^{P} \cdot p_i \cdot \min(q_{i,j,t}, 0). \end{aligned} \tag{3}$$

The first term represents revenues, the second one production costs, while the third one transportation costs. The fourth term is the overall storage costs. The function max is implemented to avoid negative inventories (i.e., backlog orders) from being counted. The last term denotes the penalty costs, which is introduced with a plus sign because stock levels would already be negative in the eventuality of unsatisfied orders. The DRL agents' goal is thus to *maximize the supply chain profit* as defined in the reward function. By design, revenues are always calculated regardless of whether the demand is effectively satisfied; however, in the event of unsatisfied orders, the penalty costs will impact the actual return for each time step in which backlog orders are counted (in the amount of the penalty coefficient).

Finally, the *state's updating rule* is defined as follows:

$$\begin{aligned} s_{t+1} = (\, &\min[(q_{0,0,t} + a_{0,0,t} - \sum_{j=1}^{J} a_{0,j,t}), c_{0,0}], \cdots, \\ &\min\left[\left(q_{I,J,t} + a_{I,J,t} - d_{I,J,t}\right), c_{I,J}\right], d_{t+1-\tau}, \cdots, d_t). \end{aligned}$$

This implies that, at the beginning of the next time step, the factory's stocks are equal to the initial stocks, plus the units produced, minus the stocks shipped. Similarly, the distribution warehouses' stocks are equal to the initial stocks, plus the units received, minus the current demand. When surplus stocks are generated, a storage cost is imposed; otherwise, a penalty cost is considered. Lastly, the demand values included in the state vector are also updated, discarding the oldest value and concatenating the most recent one.

4 Numerical Experiments

Once the environment has been specified, we implemented the agents according to three different state-of-the-art DRL algorithms: A3C, PPO, and VPG, which have been briefly introduced in Section 2. In this respect, we relied on the implementations made available by Ray [7], an open-source Python framework that is bundled with RLib, a scalable RL library, and Tune, a scalable hyperparameter tuning library. An advantage of Ray is that it natively supports OpenAI Gym. As a result, we exploited the OpenAI Gym APIs to develop the *simulator* representative of the environment and used for the agents' training process.

To assess and compare performances achieved by the adopted DRL algorithms, we also implemented a static reorder policy known in the specialized literature as the (s, Q)-policy. In our implementation, we opted to make reordering decisions independently; this means that the (s, Q)-policy parameters, $s_{i,j}$ and $Q_{i,j}$, can differ for each warehouse and product type (this policy is still defined *static* because these parameters do not change over time). To find the best possible parameters that maximize Equation (3), we developed a *data-driven approach* based on Bayesian optimization (BO). In this way, the solution method does not require making any assumptions or simplifications, and hence it is no longer problem-dependent; therefore, it can be applied to any SCIM topology and configuration just as it happens for DRL algorithms (they share, in fact, the same identical simulator).

To compare DRL and BO approaches, we also implemented an *oracle*, that is, a baseline that knows the real demand value for each product type and distribution warehouse in advance and can accordingly select the optimal actions to take a priori.

4.1 Scenarios Considered

A rich set of numerical experiments have been designed and performed to compare the performances of DRL algorithms and BO under *three different scenarios*. Each scenario is associated with different demand patterns with respect to each product type and distribution warehouse (i.e., seasonal and stochastic fluctuations). Furthermore, each scenario has different capacities and costs for evaluating in-depth the adaptability and robustness of DRL algorithms.

[7] The Ray library is available on https://www.ray.io.

Under the *one product type one distribution warehouse* (1P1W) scenario, the supply chain is set to manage just one product type. Accordingly, it consists of one factory, a factory warehouse, and one distribution warehouse; thus the input dimension of the ANN (representing the state vector) is equal to 7, given by the number of warehouses (i.e., 2, including the factory warehouse) times the number of product types (i.e., 1), plus the last demand values for each distribution warehouse and product type (i.e., 5), while the output dimension of the ANN (expressing the action vector) is 2, equivalent to the number of warehouses (including the factory warehouse) multiplied by the number of product types. Under the 1P1W scenario, which consists of five experiments (as summarized in Table 1 of the supplementary material [8]), sale prices and costs are manipulated so as to increase or decrease revenues and, consequently, the margin of return. Moreover, in the first experiment, we bound the warehouses' capacities in such a way that they are smaller than the maximum demand value (also considering the stochastic demand variation). This decision is made to study whether DRL algorithms are able to learn an efficient strategy, i.e., a strategy capable of predicting a *growing demand* and thus saving and shipping stocks in advance. Analogously, we expect a greater quantity of stocks to be stored and shipped when storage and transportation costs are low, while we expect the opposite when these costs are high. Finally, we generate multiple penalty coefficients to determine whether a hefty punishment forces DRL algorithms to be more or less effective, with particular attention to the more challenging experiments where low revenues and high costs are considered.

The *one product type three distribution warehouses* (1P3W) scenario concerns a more complex configuration, consisting of a factory, a factory warehouse, and three distribution warehouses. Even in this case, the supply chain still manages a single product type, while the input and output dimensions of the ANN are equal to 9 and 4, respectively; hence, the difficulty of the problem is increased because there is a higher number of both ANN parameters to be optimized and actions to be determined. The design of the five experiments follows that of the previous 1P1W scenario. However, a remarkable difference is found in storage capacities and costs (as depicted in Table 2 of the supplementary material [8]). In fact, we set warehouses' costs to be *directly proportional* to their corresponding capacities, that is, the less storage space we have, the more expensive it is to store a product. This scenario is also designed to investigate the DRL algorithms strategy when capacities increase, given that the search space of optimal actions grows accordingly. Furthermore, we are interested in studying how DRL algorithms react when demand, with the associated costs (i.e., production and transportation), becomes greater than actual capacities, considering that the supply chain now consists of three distribution warehouses and, consequently, the SCIM problem becomes more challenging to be tackled.

Finally, in the *two product types two distribution warehouses* (2P2W) scenario, the supply chain consists of two product types, a factory with its warehouse,

[8] The supplementary material is available on https://github.com/frenkowski/SCIMAI-Gym.

and two distribution warehouses. With this design, the number of parameters to optimize is still higher, considering that the ANN input dimension is equal to 26, while the ANN output dimension is 6. Due to computational time, we performed just three experiments under this scenario (as reported in Table 3 of the supplementary material [8]). Regarding the demand, we explore demand variations which can be different or equal, according to the specific experiment. Additionally, we thought of something different concerning storage capacities and, consequently, the search space of optimal actions. Indeed, in the last experiment, warehouses' capacities for the first product type are designed in descending order, while for the second product type in ascending order; this implies that, for example, the second distribution warehouse can store the minimum amount of stocks for the first product type and the maximum amount for the second product type. We expect that this *imbalance*, especially when combined with greater uncertainty, makes the SCIM problem more unexpected and, thus, more difficult to be effectively solved.

5 Results

To compare the performances between DRL algorithms, BO, and oracle, we simulated, for each scenario and experiment, 200 different episodes. Each episode consists of 25 time steps, and we reported the *average cumulative profit* achieved, i.e., the sum of the per-step profit at the last time step T. All experiments were run on a machine equipped with an Intel® Xeon® Platinum 8272CL CPU at 2.6 GHz and 16 GB of RAM. The hyperparameters of DRL algorithms selected for tuning have been chosen following what is presented in the Ray documentation and discussed in the papers [1,5] (they are reported in Table 4 of the supplementary material [8], along with their corresponding values). To early stop training instances associated with *bad hyperparameters configurations*, we also implemented, through Ray, the asynchronous successive halving (ASHA) scheduling algorithm [11]. It is important to note that the simulation results presented and commented in this section have been obtained by selecting, for each algorithm and experiment, the respective *best training instance* [9].

Results of numerical experiments under the 1P1W scenario are summarized in Table 1a. BO and PPO achieve a near-optimal profit in the first experiment where the demand is greater than warehouses' capacities, whereas A3C and VPG perform slightly worse. All DRL algorithms achieve comparable results in the second and simpler experiment, with higher revenues but lower transportation and penalty costs. In the third and more complex experiment, which, on the contrary, involves lower revenues and higher transportation costs and penalties, the optimal profit is relatively small, but PPO tends to behave better than other DRL algorithms. BO, PPO, and A3C obtain satisfactory profits in the fourth and more balanced experiment, with increasing revenues and maximum demand

[9] All the figures regarding the three scenario and related to the convergence and the behavior of DRL algorithms and BO are available on https://github.com/frenkowski/SCIMAI-Gym.

value but reducing uncertainty, while VPG seems to perform poorly. The main difficulty here is represented by a wider search space (caused by greater storage capacities) and higher storage costs, especially for the factory. In the fifth and last experiment, the demand uncertainty increases, the penalty costs decrease, and it is more expensive to maintain stocks at the distribution warehouse rather than at the factory, but all DRL algorithms achieve comparable and near-optimal results.

Table 1b summarizes the results for the 1P3W scenario, which in design is similar to the 1P1W scenario. The first experiment is characterized by a high maximum demand value, especially if compared with the capacities of the factory and of the first distribution warehouse; with this setting, BO performs worse than DRL algorithms. However, as PPO, it obtains a nearly optimal profit in the second experiment, where a simpler configuration is investigated. In the third and more challenging experiment, none of the algorithms achieves a profit greater than zero, with PPO achieving the worst one. Still, PPO outperforms A3C and VPG in the fourth and more balanced experiment, characterized by an increased search space and higher storage costs. Finally, BO and PPO achieve the best profits in the fifth experiment, where uncertainty and search space are increased, but fewer penalties are considered.

To conclude, Table 1c summarizes performances under the 2P2W scenario. The first experiment provides a balanced configuration, with maximum demand values and variations that change according to the specific product type, storage costs at the factory greater than those at the two distribution warehouses, and revenues particularly high for the first product type. Under such a mix, PPO achieves a good profit, as it also does A3C, which overcomes BO. For the second experiment, sales prices for the second product type are increased and, accordingly, the associated revenues grow as well. Even storage and transportation costs are decreased, while penalties increase. With this configuration, PPO still obtains a nearly optimal result, and the same happens for VPG, while BO also behaves well. In the third experiment, capacities are increased, and we design alternating storage costs; this means, for example, that maintaining stocks of the first product type at the factory warehouse is the most inexpensive option while maintaining stocks of the second product type is the most expensive. The results allow us to conclude that PPO, followed by VPG, continues to perform successfully, whereas BO seems to suffer the most.

6 Discussions and Conclusions

Results of numerical experiments demonstrated that the SCIM environment we propose is *effective* in representing states, actions, and rewards; indeed, the DRL algorithms we implemented have achieved *nearly optimal solutions* in all three investigated scenarios. In detail, PPO is the one that better adapts to different topologies and configurations of the SCIM environment achieving higher profits than other algorithms on average, although it fails to reach a positive profit in the most challenging experiment of the 1P3W scenario. VPG frequently appears

to converge to a local maximum that seems slightly distant from PPO, especially when the number of warehouses increases, but it still obtains acceptable results.

It is worthwhile to mention that the BO approach also shows remarkable results, especially when the search space of optimal actions is limited, as in the 1P1W scenario. When compared to DRL algorithms, the BO approach seems to suffer more when there are two product types or when the demand exceeds the capacities. This is mainly due to the static and non-dynamic nature of the (s, Q)-policy, which does not allow developing an effective strategy, for example, for saving stocks in advance, but, conversely, culminates in a *myopic behavior*. Nevertheless, the absence of hyperparameters to be tuned offers a considerable advantage.

Table 1. Results related to the three scenarios considered: (a) for the 1P1W scenario, it is possible to note how BO and PPO obtain near-optimal profits in general, while A3C and VPG seem more distant in terms of performance; (b) in the 1P3W scenario, PPO performs better than BO and other DRL algorithms on average, except in the third and more challenging experiment; (c) results concerning the 2P2W scenario suggest that PPO behaves well typically, whereas BO seems slightly inferior compared to the other DRL algorithms.

(a)

	A3C	**PPO**	**VPG**	**BO**	**Oracle**
Exp 1	870 ± 67	1213 ± 68	885 ± 66	**1226 ± 71**	1474 ± 45
Exp 2	1066 ± 94	1163 ± 66	1100 ± 77	**1224 ± 60**	1289 ± 68
Exp 3	−36 ± 74	**195 ± 43**	12 ± 61	101 ± 50	345 ± 18
Exp 4	1317 ± 60	1600 ± 62	883 ± 95	**1633 ± 39**	2046 ± 37
Exp 5	736 ± 45	838 ± 58	789 ± 51	**870 ± 67**	966 ± 55

(b)

	A3C	**PPO**	**VPG**	**BO**	**Oracle**
Exp 1	1606 ± 139	**2319 ± 122**	803 ± 154	486 ± 330	3211 ± 60
Exp 2	2196 ± 104	**3461 ± 120**	2568 ± 112	3193 ± 101	3848 ± 95
Exp 3	−2142 ± 128	−4337 ± 216	−2638 ± 121	**-1682 ± 196**	772 ± 21
Exp 4	−561 ± 237	**2945 ± 135**	656 ± 140	1256 ± 170	4389 ± 64
Exp 5	1799 ± 306	**2353 ± 131**	1341 ± 79	2203 ± 152	2783 ± 91

(c)

	A3C	**PPO**	**VPG**	**BO**	**Oracle**
Exp 1	2227 ± 178	**2783 ± 139**	1585 ± 184	2086 ± 173	3787 ± 102
Exp 2	1751 ± 83	**2867 ± 90**	2329 ± 98	2246 ± 114	3488 ± 63
Exp 3	1414 ± 128	**2630 ± 138**	2434 ± 156	552 ± 268	3549 ± 103

6.1 Future Research

This paper can be extended and improved in many directions as:

- Develop a *more comprehensive SCIM environment*, for example, by considering additional configurations mentioned in [9] (e.g., different demand distributions or different customers' reactions to backordering).
- Take into account the *non-linearity of transportation costs* (e.g., introducing a fixed cost independent of the number of stocks shipped effectively), as well as *non-zero leading times*.
- Use *real-world data* to validate DRL algorithms and check whether they improve the performances of currently used SCIM systems in practice.

Lastly, even the BO approach could be *extended* to other standard static reorder policies, such as the base-stock policy, which has exactly half of the (s, Q)-policy parameters and can therefore enable faster convergence times.

References

1. Alves, J.C., Mateus, G.R.: Deep reinforcement learning and optimization approach for multi-echelon supply chain with uncertain demands. In: Lecture Notes in Computer Science, pp. 584–599. Springer International Publishing (2020). https://doi.org/10.1007/978-3-030-59747-4_38
2. Boute, R.N., Gijsbrechts, J., van Jaarsveld, W., Vanvuchelen, N.: Deep reinforcement learning for inventory control: A roadmap. European Journal of Operational Research **298**(2), 401–412 (Apr 2022). https://doi.org/10.1016/j.ejor.2021.07.016
3. Chaharsooghi, S.K., Heydari, J., Zegordi, S.H.: A reinforcement learning model for supply chain ordering management: An application to the beer game. Decision Support Systems **45**(4), 949–959 (Nov 2008). https://doi.org/10.1016/j.dss.2008.03.007
4. François-Lavet, V., Henderson, P., Islam, R., Bellemare, M.G., Pineau, J.: An introduction to deep reinforcement learning. Foundations and Trends in Machine Learning **11**(3-4), 219–354 (2018). https://doi.org/10.1561/2200000071
5. Gijsbrechts, J., Boute, R.N., Mieghem, J.A.V., Zhang, D.J.: Can deep reinforcement learning improve inventory management? performance on lost sales, dual-sourcing, and multi-echelon problems. Manufacturing & Service Operations Management **24**(3), 1349–1368 (May 2022). https://doi.org/10.1287/msom.2021.1064
6. Hubbs, C.D., Perez, H.D., Sarwar, O., Sahinidis, N.V., Grossmann, I.E., Wassick, J.M.: Or-gym: A reinforcement learning library for operations research problems (2020). https://doi.org/10.48550/ARXIV.2008.06319
7. Jaakkola, T., Jordan, M.I., Singh, S.P.: On the convergence of stochastic iterative dynamic programming algorithms. Neural Computation **6**(6), 1185–1201 (Nov 1994). https://doi.org/10.1162/neco.1994.6.6.1185
8. Kemmer, L., von Kleist, H., de Rochebouët, D., Tziortziotis, N., Read, J.: Reinforcement learning for supply chain optimization. In: European Workshop on Reinforcement Learning. vol. 14 (2018)
9. de Kok, T., Grob, C., Laumanns, M., Minner, S., Rambau, J., Schade, K.: A typology and literature review on stochastic multi-echelon inventory models. European Journal of Operational Research **269**(3), 955–983 (Sep 2018). https://doi.org/10.1016/j.ejor.2018.02.047

10. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (May 2015). https://doi.org/10.1038/nature14539
11. Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Ben-tzur, J., Hardt, M., Recht, B., Talwalkar, A.: A system for massively parallel hyperparameter tuning. In: Dhillon, I., Papailiopoulos, D., Sze, V. (eds.) Proceedings of Machine Learning and Systems. vol. 2, pp. 230–246 (2020)
12. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In: Balcan, M.F., Weinberger, K.Q. (eds.) Proceedings of The 33rd International Conference on Machine Learning. vol. 48, pp. 1928–1937. PMLR, New York, New York, USA (20–22 Jun 2016)
13. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (Feb 2015). https://doi.org/10.1038/nature14236
14. Oroojlooyjadid, A., Nazari, M., Snyder, L.V., Takáč, M.: A deep q-network for the beer game: Deep reinforcement learning for inventory optimization. Manufacturing & Service Operations Management **24**(1), 285–304 (Jan 2022). https://doi.org/10.1287/msom.2020.0939
15. Peng, Z., Zhang, Y., Feng, Y., Zhang, T., Wu, Z., Su, H.: Deep reinforcement learning approach for capacitated supply chain optimization under demand uncertainty. In: 2019 Chinese Automation Congress (CAC). IEEE (Nov 2019). https://doi.org/10.1109/cac48633.2019.8997498
16. Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P.: Trust region policy optimization. In: Bach, F., Blei, D. (eds.) Proceedings of the 32nd International Conference on Machine Learning. vol. 37, pp. 1889–1897. PMLR, Lille, France (07–09 Jul 2015)
17. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms (2017). https://doi.org/10.48550/ARXIV.1707.06347
18. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al.: Mastering the game of go without human knowledge. Nature **550**(7676), 354–359 (Oct 2017). https://doi.org/10.1038/nature24270
19. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT press (2018)
20. Vinyals, O., Babuschkin, I., Czarnecki, W.M., Mathieu, M., Dudzik, A., Chung, J., Choi, D.H., Powell, R., Ewalds, T., Georgiev, P., et al.: Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature **575**(7782), 350–354 (Oct 2019). https://doi.org/10.1038/s41586-019-1724-z
21. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning **8**(3-4), 229–256 (May 1992). https://doi.org/10.1007/bf00992696
22. Wu, C., Rajeswaran, A., Duan, Y., Kumar, V., Bayen, A.M., Kakade, S., Mordatch, I., Abbeel, P.: Variance reduction for policy gradient with action-dependent factorized baselines (2018). https://doi.org/10.48550/ARXIV.1803.07246
23. Yan, Y., Chow, A.H., Ho, C.P., Kuo, Y.H., Wu, Q., Ying, C.: Reinforcement learning for logistics and supply chain management: Methodologies, state of the art, and future opportunities. Transportation Research Part E: Logistics and Transportation Review **162**, 102712 (Jun 2022). https://doi.org/10.1016/j.tre.2022.102712