Deep RL for Multi-Echelon Supply Chains

Anonymous authors
Paper under double-blind review

Keywords: Deep RL in supply chain, PPO, optimization heuristics, imitation learning

Summary

The present article studies RL methods for supply chain optimization, one of the most natural real-world applications of RL. A lot of attempts appeared during the past years in the operations research community. We approach the problem more from the RL point of view. To this end we design an abstraction that covers features of real-life supply chains typical in the process industry. Our abstraction can be implemented as a gymnasium environment to be trained with standard algorithms. It is proposed to combine optimization heuristics from operations research in combination with imitation learning to pre-train the algorithm. We compare experimentally PPO with and without pre-training to the optimization heuristic. In particular, we give a zero-shot comparison to show that deep RL agents generalize better to disruptions in the supply chain.

Contribution(s)

1. This paper proposes an abstraction of typical real-world multi-echelon supply chain problems from the process industry (e.g. the chemical or pharmaceutical industries) in the form of an RL environment. Our supply chain MDP uses order-based actions to be realistic and reduce the complexity of the action-space. We explain how to use action-shaping and -masking in different ways and apply PPO to the problem.
   Context: Without getting the attention it deserves, supply chain optimization has always been one of the prime examples for the use of reinforcement learning in real-world. Many recent articles have tackled the problem, mostly from the operations research perspective. A main caveat is often a simplified view on supply chain planning, not capturing real-world restrictions. We try to give an implementable and more realistic MDP formulation that tries to stay close to today's real-life supply chain planning.
2. The paper shows how to use classical optimization heuristics in combination with imitation learning to pre-train deep RL agents. The numerical advantage is shown on typical multi-echelon supply chain problems.
   Context: It is always desirable to start deep RL training in reasonably well-trained policies. For the supply chain problem, we do not have access to known pre-trained agents but can use optimization heuristics to create reasonable rollouts that can be fed into imitation learning algorithms to obtain reasonable policies.
3. The paper compares experimentally the deep RL to a classical planning heuristic. We show that deep RL agents can be more robust, in our experiments, they improve heuristics in zero-shot learning on changing demand.
   Context: Since the pandemic, the question of supply chain robustness gets a lot of attention. A question of large practical importance is the understanding of robustness of planning algorithms used in suddenly changed environments, such as suddenly increased or decreased demand. We show experimentally that deep RL agents are more robust than typical planning heuristics that are run in standard supply chain planning software.

Deep RL for Multi-Echelon Supply Chains

Anonymous authors
Paper under double-blind review

Abstract

The present article studies RL methods for multi-echelon inventory optimization, one of the most natural real-world applications of RL. A lot of attempts appeared during the past years in the operations research community; we approach the problem from the RL point of view. To this end, we design an abstraction that covers features of real-life supply chains typical in the process industry. Our abstraction can be implemented as a gymnasium environment to be trained with standard algorithms. We propose to combine MRP optimization heuristics from operations research in combination with imitation learning to pre-train the RL algorithms. We compare experimentally PPO with and without pre-training to the MRP heuristic. In particular, we give a zero-shot comparison to show that deep RL agents generalize better to disruptions in the supply chain.

1 Introduction

The field of deep RL has seen remarkable success in various fields. Breakthroughs range from games Silver et al. (2016), over the optimization of fusion reactors Degrave et al. (2022), to various topics in the training of LLMs. A field that has seen relatively little progress given its tremendous industry importance, is multi-echelon (also called multi-level) optimization, see e.g. Barbosa-Povoa et al. (2017). In supply chain optimization, decisions are made (in real life by teams of supply chain planners) that range from the procurement of materials, over manufacturing steps, to the distribution in a logistic network. In this article, we are interested in problems that are typical for the process manufacturing industry (e.g. the chemical or pharmaceutical industries). Process industry have the challenge of manufacturing ingredients via multiple processing, with long lead times, shared and constraint resources, with typically global supply via 1-2 plants.

A typical real-world situation is displayed in Figure 1. The example has a single factory and a number of warehouses distributed over several continents. Customer demand is fulfilled only by gray warehouses, while white warehouses are inter-company distribution centers. The lead times in replenishment vary a lot depending on inter-continental or local replenishment.

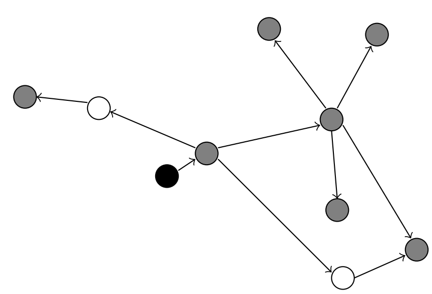


Figure 1: Real-world logistics network over continents. Black is a factory, gray warehouses have customer demand, and white intercompany distribution centers.

The topic we are interested in is *multi-echelon inventory optimization (MEIO)*. How to make optimal manufacturing/replenishment decisions that fulfill customer demand at minimal cost (storage, shipment, and backorder costs). In this article, we introduce an order-based RL framework that imitates real-world supply chain planning by placing orders into an order book. Our framework is implemented as a gymnasium environment to be trained with standard deep RL algorithms. It turns out that training the environments is delicate, fast PPO training requires a reasonable initial policy to avoid metastability effects. We introduce a new trick to deep RL training in supply chain optimization by combining deep RL and standard optimization heuristics. Using some heuristic (good or bad), we use imitation learning to imitate a policy network that is used to initiate the PPO training. We show experimentally that pre-training through imitation improves the training a lot. Previous work on deep RL in supply chain optimization has shown that beating known rule-based optimization heuristics is not easy, see e.g. Gijsbrechts et al. (2022). This is similar for our order-based MDP problem. Still, learning (near) optimal policy networks has the big advantage that the evaluation is much cheaper compared to rule-based heuristics and additionally one can expect generalization effects of the neural networks. The latter might turn out powerful for future deep RL applications in supply chain problems as recent years have shown that disruptions in the supply chain cause major challenges, more robust supply chain planning is key.

Related work: Despite the challenges, industry solutions and a huge body of operations research literature give approximate solution methods to various optimization problems in inventory management, production planning, and logistics operations. Problems often fall into the category of NP-hard problems, where finding optimal solutions becomes computationally infeasible as the scale and intricacy of the supply chain increase (Gayon et al., 2017). Consequently, traditional systems rely on heuristic algorithms to provide approximate solutions (Marklund & Rosling, 2012; Zhao & Zhang, 2020). While many methods offer computational efficiency and simplicity, they are limited by their simple nature, they tend to struggle to solve the optimization problems if the complexity is high, and they do not generalize to changing conditions. There is a growing body of research towards deep learning and RL for supply chain optimization. Q-learning was used, for instance, for the beer game (Oroojlooyjadid et al., 2022), PPO was used, for instance, on a small toy problem in Vanvuchelen et al. (2020). Inventory management for a single-node problem has been studied in Qi et al. (2023). Hubbs et al. (2020) provide a gym environment for simpler supply chain problems. Perez et al. (2021) trained PPO and compared it to perfect information oracles. A comparison of A3C with classical operations research methods has been provided in Gijsbrechts et al. (2022).

In contrast to the present articles, all these articles have in common that their focus is less on the specifics of RL training on supply chain problems but much more on the comparison to different heuristics from operations research. For a recent article that contributed to the RL specifics of (rather different) supply chain problems we refer to Madeka et al. (2022).

Main contributions:

- **Modeling:** An order-based MDP formalism is set up with main features of industry supply chains.
- **RL techniques:** Action-shaping and -masking are used to deal with huge action-spaces. Imitation learning is used to leverage simple optimization heuristics for PPO training.
- **Experiments:** Different learning strategies are compared experimentally for learning efficiency, cost vs. backorder performance, and zero-shot generalization to demand changes.

2 Modeling order-based supply chain environments

As a first approximation for a multi-echelon supply chain problem, the reader might want to think of a multi-graph G in which nodes represent material-storage-locations (different materials at the same location are different nodes) and edges their abstract relations. In a logistics network for one material, the nodes might just represent warehouses, in manufacturing, nodes might also be different materials at the same location. Our modeling allows both. A relation could be a shipment from one location to another, but also a production step involving multiple materials manufactured into a new material at the same location. With regards to real-world supply chain planning we distinguish between *procurement*, buying materials from suppliers, *manufacturing*, producing new materials from other materials, and *replenishment*, transporting materials between storage locations. To have a joint modeling of actions we introduce the concept of *value creation* objects. These are supply chain planning steps that add value to the supply chain. A value creation is based on a labeled finite weighted sub-graph of the supply chain graph G and performs a typical supply chain step. This can be a shipment that only involves two nodes connected by an edge, or a production step that involves a sub-tree of G, where several materials are manufactured into a new material. Value creations v are executed through orders. If the order is feasible, the value creation specifies the quantities of materials to be shipped or produced. Quantities are measured in integer multiples of a fixed lot size L_v. Each value creation has an associated (random) lead time τ_v, the time between placing the order and receiving the material. We assume that materials are consumed immediately at the source nodes, become available at the destination node after the lead time τ_v at a cost c_v per lot size. Required real-world information needed for this setup is available in ERP (enterprise resource planning) systems, more precisely from master data such as BOM (bill-of-materials) files or from historic data. Each node n maintains an inventory level $I_{n,t}$ at time t, representing the quantity of stored materials. For demand nodes (e.g. customers can buy at these locations), random customer demand $d_{n,t}$ specifies the demand at time t. When $I_{n,t} \geq d_{n,t}$, demand is fully met and inventory decreases accordingly. If $d_{n,t} > I_{n,t}$, the unmet amount $B_{n,t}$ is called a backorder. Both inventory levels and backorders are dynamically updated based on sequential order placements that trigger value creations.

Supply chain environment: To create an MDP that models the order-based decision making in supply chain planning a number of quantities need to be defined. In the reality, most data from the table are either known from master data or can be estimated from historic data.

Symbol	Description
N	Set of all material-storage-locations.
V	Set of value creations, a set of weighted subgraphs of the complete graph with N nodes. The edge weights determine the portions of each ingredient.
d_v	Destination of value creation v, the node that receives the material
s_v	Set of sources of value creation v, nodes that contributed to the value creation v
c_n^h	Storage cost per time unit and unit of material stored at node n.
c_n^b	Backorder cost per time unit and unit of material stored at node n.
w_n	Maximum inventory capacity at node n.
$\tilde{w}_n$	Maximum allowed backorder at node n.
c_v	Cost per lot for orders of a value creation v.
q_v	Lot size for value creation v.
$g_{v,n}$	Input material quantity from node n per one lot size of value creation v.
$k_v^{\max}$	Maximum number of lots per order for a value creation v.
$\mathcal{D}_n$	Demand distribution for material at node n.
$\mathcal{Q}_v$	Lead time distribution on value creation v.
$I_{n,t}$	Inventory level at node n at time t.
$B_{n,t}$	Backorder level at node n at time t.
$D_{n,t}$	Demand for material at node n at time t (distributed according to $\mathcal{D}_n$).
L_v	Number of lots created in value creation v.
O	Set of orders $o = (v, t, \tau, L)$, where v is a value creation, t the placement time, τ the lead time (distributed according to $\mathcal{Q}_v$), and L the number of lot sizes.

Table 1: Definitions for constants, distributions and variables.

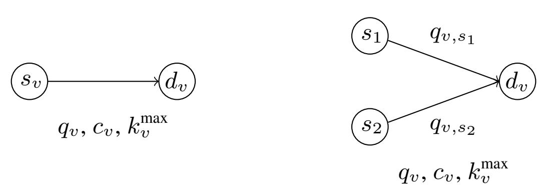


Figure 2: Schematic description of value creations. Left: Replenishment v with lot size q_v, cost c_v, maximal replenishment capacity $k_v^{\max}$. Right: Manufacturing step using materials s_1, s_2 requiring quantities q_{v,s_1}, q_{v,s_2} to create a lot of size q_v at cost c_v. $k_v^{\max}$ lots can be processed at once.

We define the problem as a discounted Markov Decision Process (MDP) with states, actions, and rewards. An MDP is defined by a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, P is the transition probability function, R is the reward function, and γ is the discount factor. The goal is to maximize the expected discounted total reward $\max_\pi \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$ over all policies (kernels on $\mathcal{A} \times \mathcal{S}$). Defining an MDP for supply chain problems is non-trivial, in particular caused by the complexity of allowed actions. An example to take into account is a planner that can only ship as much stock as is available.

States: The state-space is $\mathcal{S} = \mathbb{R}^{|N|} \times \mathbb{R}^{|N|} \times O^\infty$, where we denote by O^∞ vectors of orders (triplets) of arbitrary finite length. States represent inventory levels and backorder levels for each node n and the current order book. The order book contains a finite number of tuples with start time, lead time, and number of lots demanded for a value creation.
All actions: The formal action-space is $\mathcal{A} = \mathbb{N}^V$. A finite action vector a represents the lot sizes of all action creations asked by the RL planner to be placed. If an action a is performed at time t, then new orders (v, t, τ, L) will be placed in the order book and used to update the environment.
Feasible actions: Not all actions are allowed, actions are only allowed when they are possible. For example, a shipment order can only be placed if material is available at the source. The feasible set of actions is defined by three constraints: resource capacity, order capacity and material availability.

- *Available manufacturing capacity:* Manufacturing orders can only be placed if the previous order o of the same value creation is processed, i.e. $t_o + \tau_o < t$. There is no such constraint for procurement and replenishment, new shipment orders are always possible.
- *Order capacity:* There is a maximal number $k_v^{\max}$ of lots that can be manufactured/replenished/procured. For replenishment e.g. the maximal number of containers on a vessel.
- *Material availability:* The sum over all value creations placed by the action a must satisfy

$$\sum g_{v,n} L_v \leq I_n$$

for all nodes n. The constraint asks that all orders added to the orderbook can be satisfied immediately from the current inventory.

Environment dynamics: The environment runs in discrete time-steps by fulfilling orders from the order book O that is filled by the actions. To see how states transition we need to identify new inventory and backorder values, as well as the change in the order book. Changes in the order book are simple. New orders are included, finished orders (i.e. $t_o + \tau_o < t$) are removed. To describe the inventory and backorder update, we keep track of a single inventory variable (instead of inventory and backorder separately) that can have negative values, called the generalized inventory level. The generalized inventory level at time t at node n is denoted by $G_{n,t} \in \mathbb{R}$. This is equal to either the inventory level $I_{n,t}$ when it is positive or to the negative backorder $-B_{n,t}$ when it is negative. The update rule for generalized inventory values is as follows:

$$G_{n,t} = G_{n,t-1} + \underbrace{\sum_{o \in O: d_{v_o} = n} q_{v_o} L_{v_o} \mathbf{1}_{t_o + \tau_o = t}}_{\text{goods received by orders finished at time } t} - \underbrace{\sum_{o \in O: n \in s_{v_o}} g_{v_o,n} L_{v_o,t_o} \mathbf{1}_{t_o = t}}_{\text{goods issued to satisfy orders}} - \underbrace{D_{n,t}}_{\text{demand}}, \tag{1}$$

where the random demand $d_{n,t}$ is sampled from an iid distribution $d_{n,t} \overset{\text{i.i.d.}}{\sim} \mathcal{D}_n$ at each time t. In the update formula $\mathbf{1}$ denotes the indicator function. $G_{n,t}$ is then clipped to the range $G_{n,t} \in [-\tilde{w}_n, w_n]$ to enforce the storage capacity and backorder capacity conditions. While the maximum inventory constraint w_n is relevant to model maximum storage capacities, the maximal backorder constraint $\tilde{w}_n$ is mostly technical and will be set to a large number. Changes in generalized inventory level result from three factors: goods received from finished orders, goods issued due to started orders, and goods issued due to sales towards customers. In this model, unfulfilled demand is not erased, but is instead stored as backorder, eventually converting into fulfilled sales once inventory becomes available to clear the backlog. The assumption is unrealistic in B2C (business-to-customer) models but common in B2B models where customers cannot change easily their suppliers for regulatory reasons. Inventory level and backorder are deduced from $G_{n,t}$:

$$I_{n,t} = \max(0, G_{n,t}) \quad \text{and} \quad B_{n,t} = \max(0, -G_{n,t}). \tag{2}$$

Reward function: For state-action pairs (s, a) described above, the reward is defined as

$$R(s,a) = -\sum_{n\in N}\Big(\underbrace{c_n^h\, I_n}_{\text{inventory holding cost}} + w\underbrace{c_n^b\, B_n}_{\text{backorder cost}}\Big) - \sum_{v\in V}\underbrace{c_v\, L_v}_{\text{value creation execution costs}}, \tag{3}$$

where inventory/backorder values are from the state and the lot sizes from the action. We introduce a *backorder weight* $w > 0$ that is not part of the model. This non-trivial weight allows the decision maker to adjust preferences between inventory/backorder (see Figure 5). We use $w = 30$ in our experiments (justified by the Pareto curve in Figure 5).

3 RL training

Training multi-echelon supply chain environments is challenging for a number of reasons. Other than the challenge of creating an efficient simulator, the action-spaces are too big for naive exploration to succeed. The action-space of a multi-echelon supply chain can grow exponentially with the size of the supply chain network, making it extremely hard for an RL agent to find a good policy through unguided exploration. To enable deep RL training with PPO, we built a gymnasium environment for the order-based supply chain management described above. To get hold of the large action-space, we introduce an action-shaping and -masking component. Finally, we propose a new pre-training concept for RL problems with known optimization heuristics.

3.1 Action-shaping and -masking

While our order-based modeling already gives a lot of structure and reduction in size to the action-space, two more ingredients are used.

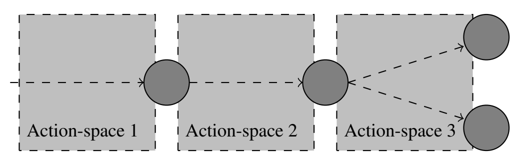


Figure 3: Independent action-spaces, dotted lines represent parts of the supply chain.

Multi-discrete action shaping: In the order-based modeling above, we introduced the concept of value creations to structure dependent sub-actions in the supply chain. Value creations allow us to simplify the action-space enormously, as not all combinations of actions at all nodes must be taken into account when placing an order. Additionally, we use multi-discrete action shaping, see Kanervisto et al. (2020). The illustration shows that we do not need to consider all combinations of value creation orders possible, but only those with dependencies through their sources. Structuring the actions into independent components and utilizing multi-discrete action-spaces, we improve computational efficiency.

Invalid action masking: A substantial portion of actions is typically invalid due to constraints such as limited material availability. If no products exist in the supply chain, production must precede any other action, rendering nearly all actions initially invalid. We utilize invalid action masking, see e.g. Huang & Ontañón (2020), in the policy network that outputs probabilities for all actions. To implement masking, we adjust the logits before applying a final softmax function. Invalid actions are assigned a large negative number to ensure invalid actions have near zero probabilities in the softmax output.

3.2 Reward optimized MRP heuristic

Material Requirements Planning (MRP) is a dynamic programming inspired rule-based heuristic used to determine the quantity and timing of production and replenishment orders, more details in the Appendix 7 for a very simple variant of MRP. Based on a number of checks, MRP suggests times and quantities for value creations. It can be interpreted as a policy which is inefficient to evaluate and is *not* designed to maximize a particular reward function. What makes the MRP algorithm unpleasant is not only the effort in computing the action but also that the algorithm requires so-called safety stock levels S_n as a hyperparameter. For our experiments, we use a novel safety stock approach that is promising in its own rights, as it can be used for complex supply chains. This is in contrast to safety stock policies (such as based on Gaussian tail bounds for estimated one-step workloads) used in real-life systems that are optimal in very small examples but can fail badly for complex systems. Interpreting safety stocks as a hyperparameter to the "MRP policy" we use Bayesian optimization (scikit-optimize library) to maximize the value function $V(S) = \mathbb{E}_{\text{MRP}(S)}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$. The approach can be seen as a combination of standard MRP and reward optimization. The maximal safety stock vector is denoted by S^*, the *reward optimized MRP policy* by MRP(S^*).

3.3 Online imitation learning

We will show below that standard RL training using PPO is delicate for multi-echelon supply chain problems and propose two fixes. The most effective one is to imitate an MRP heuristic and use the imitated policy network as a pre-training for PPO. For the imitation we use DAgger (dataset aggregation), a popular imitation learning algorithm for online learning (Ross et al., 2011). DAgger is an improvement on Behavior Cloning (BC) (Pomerleau, 1991) that replaces the static dataset of BC with a dynamic dataset, containing trajectories generated by the training agent and the expert together and annotated by the expert. DAgger training consists of N iterations, generating new training data each iteration while increasing the influence of the learned agent on the generated trajectories. At iteration t, a mixed policy π_t is used to generate environment interactions:

$$\pi_t = \begin{cases} \pi_* & : \text{with probability } \beta_t \\ \pi_{\theta_i} & : \text{with probability } 1 - \beta_t \end{cases},$$

where $\beta_t \in [0, 1]$ is a mixing parameter that controls the probability of following the expert policy π_* versus the learned policy π_{θ_t}. Here π_θ is a policy parametrized by a neural network with fixed architecture and weight vector θ. The collected dataset $\mathcal{D}_i = \{(s, \pi_t(s))\}$ of visited states and corresponding actions is added to an aggregated dataset: $\mathcal{D} = \bigcup_{i=1}^{N} \mathcal{D}_i$ and a new policy $\pi_{\theta_{i+1}}$ is then trained on $\mathcal{D}$. We follow the typical method of linearly annealing β_i from 1 to 0.

3.4 Training

Standard PPO: Training is performed using PPO with random (Glorot) initialization. We use the maskable-PPO implementation of Stable Baselines3 available in SB3-Contrib (Raffin et al., 2021), see appendix for details. It turns out that the training is extremely slow, multi-echelon supply chain training is very much prone to metastability effects (see appendix). What happens is the following: the agent learns quickly to optimize all nodes except the last, where all material exceeding the storage capacity is discarded. Leaving this local maximum is delicate for gradient-based algorithms. To learn to optimize the terminal node, earlier sub-decisions become suboptimal so the algorithm must first worsen before improving. Orange learning curves in the experiments below show the metastability, the agent is stuck in suboptimal strategies for millions of interactions and suddenly improves (when inventory of the last node is decreased).

Modified reward function: A first mitigation of the metastability problem is to modify the reward function to penalize the waste of excess material. We change the reward function R into

$$\bar{R}(s, a) = R(s, a) - c \sum_{n \in N} c_n^h \cdot e_{n,t}, \tag{4}$$

where $e_{n,t}$ is the amount of excess material discarded from node n at time t when it exceeds the capacity w_n. c is a penalty weight, set to 100 in the experiments. We note that the change of R to $\bar{R}$ does not change the optimal solution, since the capacities w_n are typically much higher than the optimal inventory levels. However, our experiments show that the use of $\bar{R}$ speeds up training significantly and allows the agent to escape suboptimal local maxima.

MRP pre-trained PPO: We now propose a novel approach to drastically improve PPO training. The idea is to make the PPO agent start in a favorable parameter valley, a valley that represents policies that do not sacrifice optimality at single nodes (typically the last). For the pre-training we imitate the MRP(S) policy. We learn a neural network that mimics the same action making as MRP(S). Every evaluation of the MRP(S) policy is costly, the entire rule-based algorithm needs to be performed. We run DAgger for ten iterations only, generating a single trajectory of 3600 (10 years) time-steps in each iteration. For the ith iteration of DAgger the expert policy $\pi_* := \text{MRP}(S)$ is chosen in every step with decreasing probabilty β_i. This is enough to bring the policy above 80% accuracy predicting MRP's actions. Even though the evaluation of the MRP policy is costly, a few thousand evaluations are neglegible compared to millions of iterations needed for PPO. The MRP pre-trained policy is then used to initialize the PPO training. It is important to note that the choice of safety-stock vector S is not crucial, every reasonable choice improves the PPO training a lot.

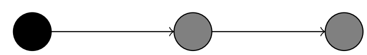
Algorithm 1 MRP pre-trained PPO for multi-echelon inventory optimization

Step 1 (Safety stocks): Determine good safety stock vector S, e.g. using Bayesian opt.
Step 2 (Imitation learning): Perform DAgger using expert policy $\pi_* = \text{MRP}(S)$:
for for $i = 1, ..., N$ **do**
  Obtain rollout using in every step $\text{MRP}(S)$ (resp. π_{θ_i}) with probability β_i (resp. $(1 - \beta_i)$).
  Add rollout state-action pairs to buffer $\mathcal{D}$, use supervised imitation on $\mathcal{D}$ to obtain θ_{i+1}.
end for
Step 3 (PPO): Run PPO with initial policy network obtained in Step 2. =0

4 Experiments

The algorithms are compared on a simple three-nodes example and the more complex ten-nodes example from Figure 1.

The simple problem consists of a factory (black) and two warehouses (gray) with random customer demands (negative binomial with means 367 (middle) and 172 (right)). Replenishment from factory to warehouse and from warehouse to warehouse have random lead times (Poisson auto-regressive with means 6 (resp. 15)). Variants of the simple toy example are often used in the supply chain literature. They also appear in industry, parameters for our experiment (see Appendix) stem from a real-world problem in the process manufacturing industry.



Example, gray nodes have demand

To show that the choice of S is not crucial to have a well-performing MRP pre-trained PPO agent we used a decent safety stock vector S instead of the optimal S^*.

Learning curves: MRP value functions are plotted as horizontal lines. RL agents are evaluated every $2.5 \cdot 10^4$ environment interactions, value functions are evaluated on 80 episodes of length 3600 (representing ten years). The training curves are smoothed with a Gaussian filter with $\sigma = 2$. Standard PPO struggles to learn the optimal policy. Further investigation of single node inventories shows that the algorithm is stuck in local maxima of strategies that sacrifice the final warehouse which is kept at maximal capacity. The PPO agent takes millions of interactions to find a parameter region that reduces the inventory at the final warehouse. PPO with modified rewards is much worse at the beginning (due to the penalization) and then improves much quicker, as expected the metastable behavior dissapeared. MRP pre-trained PPO clearly beats PPO without pre-training.

Performance - cost vs. backorder: In Figure 5 we provide experiments to see the effects on cost vs. backorder when varying inventory/backorder preferences. The graphs show that MRP pre-trained PPO agents slightly outperform the reward-optimized MRP policy. It is natural to ask why it might be interesting to prefer RL policies to the simple (reward optimized) MRP policy. First, it is timely to evaluate for every decision the rule-based MRP policy, see Appendix 7. In contrast to evaluating a neural network for RL policies, it requires a number of algorithmic steps that in real-life large supply chains is an important problem. Secondly, as our next result shows, the RL agent generalizes better to changes in the environment, one of the key topics in current supply chain organizations.

Robustness: In Figure 6 we provide zero-shot experiments, where the trained policies are compared on unseen demand distributions with changed customer demand. For decreased demadn the RL agent generalizes better than reward-optimized MRP, the inventory/backorder situation improves.

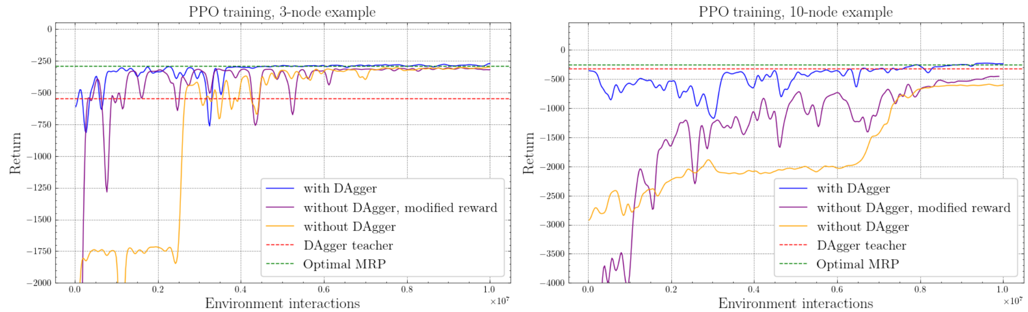


Figure 4: Learning curves for three PPO trained agents compared to the (constant) value function of MRP agents (green with reward optimized S^*, red with suboptimal S used for DAgger training). DAgger pre-training (blue) enables better performance than optimized MRP. Without pretraining, adding the capacity penalty of Eq. (4) helps to avoid overstocking material (purple vs. orange).

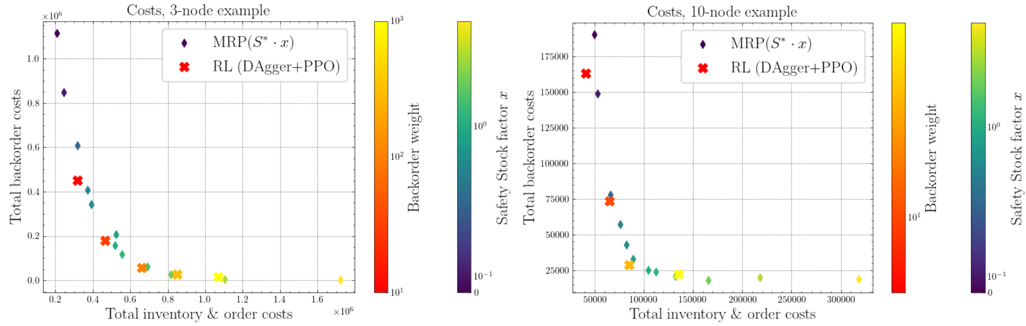


Figure 5: Pareto curves (backorder vs. inventory costs) of RL agents and MRP. We fix all RL training parameters but scale the backorder weight w of Eq. (3). Similarly, for MRP we change inventory/backorder preferences by scaling all safety stock values a factor x.

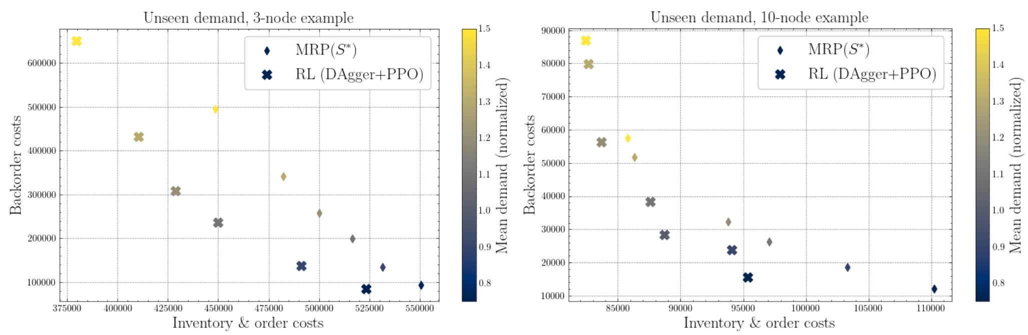


Figure 6: Zero-shot performance for new demand. Changing magnitude of demand, for lower demand RL agent generalizes clearly better.

5 Conclusion

In the present article we introduced an MDP modeling for order-based supply chain management. Combining action-shaping and -masking to reduce the action-space we created a gymnasium environment to run RL algorithms. Since plain vanilla PPO is slowed down by metastability effects we introduced supply chain specific patches, in particular using imitation learning on a rule-based MRP heuristic. The approach has potential to be used in other RL approaches to classical OR problems.

Due to its enormous importance in supply chain management it would be very beneficial in future work to understand how policy networks can be improved on zero-shot learning by training the agent on more and different extreme scenarios. The generalization ability of neural networks has a potential huge benefit to supply chain robustness for sudden changes in real-life.

References

Ana Barbosa-Povoa, Albert Corominas, and João Miranda. *Optimization and Decision Support Systems for Supply Chains*. 01 2017. ISBN 978-3-319-42419-4. DOI: 10.1007/978-3-319-42421-7.

Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego Casas, Craig Donner, Leslie Fritz, Cristian Galperti, Andrea Huber, James Keeling, Maria Tsimpoukelli, Jackie Kay, Antoine Merle, Jean-Marc Moret, and Martin Riedmiller. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602:414–419, 02 2022. DOI: 10.1038/s41586-021-04301-9.

Jean-Philippe Gayon, Guillaume Massonnet, Christophe Rapine, and Guy Stauffer. Fast approximation algorithms for the one-warehouse multi-retailer problem under general cost structures and capacity constraints. *Mathematics of Operations Research*, 42(3):854–875, 2017.

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

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Shengyi Huang and Santiago Ontañón. A closer look at invalid action masking in policy gradient algorithms. *arXiv preprint arXiv:2006.14171*, 2020.

Christian D Hubbs, Hector D Perez, Owais Sarwar, Nikolaos V Sahinidis, Ignacio E Grossmann, and John M Wassick. Or-gym: A reinforcement learning library for operations research problems. *arXiv preprint arXiv:2008.06319*, 2020.

Anssi Kanervisto, Christian Scheller, and Ville Hautamäki. Action space shaping in deep reinforcement learning. In *2020 IEEE conference on games (CoG)*, pp. 479–486. IEEE, 2020.

Dhruv Madeka, Kari Torkkola, Carson Eisenach, Anna Luo, Dean P Foster, and Sham M Kakade. Deep inventory management. *arXiv preprint arXiv:2210.03137*, 2022.

Johan Marklund and Kristoffer Rosling. Lower bounds and heuristics for supply chain stock allocation. *Operations Research*, 60(1):92–105, 2012.

J. Orlicky. *Material requirements planning: the new way of life in production and inventory management*. New York: McGraw-Hill, 1975.

Afshin Oroojlooyjadid, Mohammad Nazari, Lawrence V. Snyder, and Martin Takáč. A deep q-network for the beer game: Deep reinforcement learning for inventory optimization. *Manufacturing & Service Operations Management*, 24(1):285–304, 2022.

Hector D Perez, Christian D Hubbs, Can Li, and Ignacio E Grossmann. Algorithmic approaches to inventory management optimization. *Processes*, 9(1):102, 2021.

Dean A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural computation*, 3(1):88–97, 1991.

Meng Qi, Yuanyuan Shi, Yongzhi Qi, Chenxin Ma, Rong Yuan, Di Wu, and Zuo-Jun Shen. A practical end-to-end inventory management model with deep learning. *Management Science*, 69 (2):759–773, 2023.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Nathalie Vanvuchelen, Joren Gijsbrechts, and Robert Boute. Use of proximal policy optimization for the joint replenishment problem. *Computers in Industry*, 119:103239, 2020.

Min Zhao and Mingyu Zhang. Multiechelon lot sizing: New complexities and inequalities. *Operations Research*, 68(2):534–551, 2020.

Supplementary Materials

The following content was not necessarily subject to peer review.

6 Experimental details

6.1 Environment

Here are the paramters for the 3-node example, the exact parameters for the 10-node example must be kept confidential.

node	c_n^h	c_n^b	w_n	$\tilde{w}_n$	demand
middle	26.7	34	170093	12465	$\mathrm{NB}(0.11, 0.0003)$
right	29.5	42	127646	8025	$\mathrm{NB}(0.081, 0.00047)$
replenishment	q_v	$k_v^{\max}$	$k_v^{\max}$		lead time
left	4180	2	0		$\mathrm{ARPois}(6, 20, 0.98)$
right	4100	2	1500		$\mathrm{ARPois}(15, 20, 0.98)$

Both demand distributions are Negative Binomial, $\mathrm{NB}(r, p)$. To reflect realistic fluctuating lead times, we use an autoregressive variant of the Poisson distribution, $\mathrm{ARPois}(\lambda_0, h, \phi)$. This distribution produces each time-step t a lead time τ_t which is drawn from the distribution $\mathrm{Pois}(\lambda_t)$, where λ_t depends on the previously h drawn lead time values τ according to the formula:

$$\lambda_t = \max\left(0, \phi \cdot \frac{\sum_{t' \in [t-h, t]} \tau_t'}{h} + (1-\phi)\lambda_0\right) \tag{5}$$

This produces a time-correlated Poisson distribution which retains an expected value of $_0$.

6.2 Details on PPO training

We train feedforward neural nets with PPO. We add skip connections (He et al., 2016) every two layers to enable training deep networks, effectively using 2 residual blocks in both the value and policy networks.

Hyper-parameter	parameter value
no. of environment interactions	10^7
policy network	width 256, depth 4
actor network	width 256, depth 4
activation function	ReLU
discount factor γ	0.99
GAE paramter	0.95
Adam learning rate	$2.5 \cdot 10^{-4}$
batch size	64

6.3 Metastability

The learning curves in Figure 4 highlight the difficulty of deep RL training for multi-echelon supply chain optimization if the reward function is not chosen carefully. Metastability effects occur as there

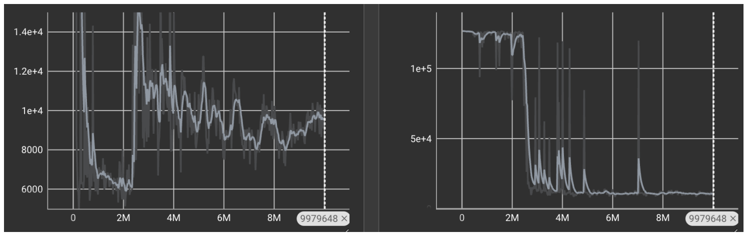


Figure 7: 3-node example, inventory during training at middle node (left) and right node (right)

are suboptimal strategies that manifest local maxima for the parameter vectors of the policy network. As a consequence, gradient based algorithms struggle to improve the suboptimal strategy. To explain what happens we plotted the inventory levels of the 3-node example during training (inventory plots look similar for the 10-node example). The learning curve of Figure 4 shows a sudden improvement after 2.5M environment interactions, preceded by a return drop. This is reflected in Figure 7. The agent finds quickly the suboptimal strategy to optimize the inventory at the middle node while completely sacrificing the terminal right node, running at maximal inventory 127646. At 2.5M iterations the RL agent deviates from the local maximum and explores a new strategy, reducing inventory at the terminal node and increasing inventory at the middle node.

7 Material requirements planning (MRP)

The main algorithmic invention of this article is to combine off-the-shelf RL training (PPO) through imitation learning with rule-based heuristics from operations research (OR). There are several OR algorithms to solve approximately different supply chain problems. For the multi-echelon inventory optimization (MEIO) problem studied in the present article we use a dynamic programming inspired rule-based algorithm that is (with various modifications) implemented in many industry supply chain solutions. We now give a quick overview for the interested RL researcher.

The rule-based algorithm implements a time-phased Material Requirements Planning (MRP I) system to maintain inventory levels above safety stock thresholds across all nodes in the supply chain. Rooted in the foundational work Orlicky (1975), the process begins by exploding dependencies from downstream nodes (e.g., retailers or finished goods) to upstream suppliers, following the hierarchical structure of the multi-echelon supply chain. Inventory projections are calculated in daily time buckets over a fixed $H = 150$ planning horizon. Starting from the current day t, the system computes the projected available balance (PAB) for each subsequent day $s \in [t, t+H]$, accounting for scheduled receipts, planned orders, and demand forecasts. If the PAB is projected to fall below the safety stock level at time T, a planned order is generated to replenish the deficit. Orders are offset by lead times using backward scheduling: for an order requiring τ days of lead time, the release date is set to $T - \tau$. If this calculated release date precedes the current day t, the order is flagged as overdue and scheduled for immediate release. This daily recalibration ensures alignment with the core principle of time-phased net requirement calculation, where material plans are dynamically adjusted to reflect real-time demand and supply conditions. Rule-based MRP algorithms are dynamic-programming, heuristic-based algorithms. It implements a safety-stock approach to managing the supply chain, meaning it predicts the future inventory levels of all nodes in the chain and tries to ensure inventory never falls below the "safety stock" that must be given to the algorithm.

Since we use the MRP algorithm in our examples without multi-material manufacturing steps, we give pseudo-code for a simplified version of MRP. It should be noted that the algorithm is a very simple MRP variant that does not estimate demand expectations and lead times on the run. We do this for a fair comparison to the RL agents, otherwise demand distributions should also be included in the MDP state-space and not be given as part of the model.

There are two novel ideas we add on the standard OR literature.

Algorithm 2 MRP Algorithm (without multi-material manufacturing steps)

Input: expected demands $\mathbb{E}(d)$ and lead times $\mathbb{E}(l)$, safety stocks S_n for all nodes, current generalized inventories $G_{n,t}$ and running orders set O.
for each node n in topological order **do**
  for each time-step $s \in [t, t+H]$ **do**
    $gen(n, \tau) \leftarrow$ amount of additional material by finished orders
    $G_{n,s} \leftarrow G_{n,s-1} + gen(n, s) - \mathbb{E}\left(d(n)\right)$
    if $G_{n,s+1} < S_n$ **then**
      Set number of lots L to minimal number containing at least amount $S_n - G_{n,s+1}$.
      if procurement is possible **then**
        Add to O an order from a source node. L lots, start time: $\max(t, s - \mathbb{E}(l))$
      else
        Choose source node n' that maximizes $\frac{G_{n',s+1}}{\text{num\_outgoing}(n')}$, where $\text{num\_outgoing}(n')$ denotes the number of nodes supplied by node n'.
        Add to O an order from node n' to node n. L lots and start time $\max(t, s - \mathbb{E}(l))$
      end if
    end if
  end for
end for
Output: all orders in O that start at time t. =0

- We interpret MRP(S) $=: \pi$ as a policy. The action (orders) in the state S (inventory level and current order book) are the output orders of the algorithm given above (the orders suggested by the algorithm to be placed at initial time t).
- The safety stock vector S is a required input to the algorithm. We define the reward-based optimal safety stock vector S^* by maximizing the expected reward R under the MRP run defined by the MRP algorithm: $V(S) = \mathbb{E}_{\text{MRP}(S)}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$. Since S is a hyperparameter to the algorithm, it is natural to use a Bayesian optimization algorithm to do so.