Math Programming based Reinforcement Learning for Multi-Echelon Inventory Management

Pavithra Harsha, Ashish Jagmohan, Jayant R. Kalagnanam, Brian Quanz, Divya Singhvi

1 IBM Research, Thomas J. Watson Research Center, Yorktown Heights, NY 10598, USA
2 Stern School of Business, New York University, New York, NY 10012, USA
{pharsha,ashishja,jayant,bqliu,ds6844}@us.ibm.com, ds6844@stern.nyu.edu

Abstract

Reinforcement learning has lead to considerable breakthroughs in diverse areas such as robotics, games and many others. But the application to RL in complex real-world decision making problems remains limited. Many problems in operations management (inventory and revenue management, for example) are characterized by large action spaces and stochastic system dynamics. These characteristics make the problem considerably harder to solve for existing RL methods that rely on enumeration techniques to solve per step action problems. To resolve these issues, we develop Programmable Actor Reinforcement Learning (PARL), a policy iteration method that uses techniques from integer programming and sample average approximation. Analytically, we show that the for a given critic, the learned policy in each iteration converges to the optimal policy as the underlying samples of the uncertainty go to infinity. Practically, we show that a properly selected discretization of the underlying uncertain distribution can yield near optimal actor policy even with very few samples from the underlying uncertainty. We then apply our algorithm to real-world inventory management problems with complex supply chain structures and show that PARL outperforms state-of-the-art RL and inventory optimization methods in these settings. We find that PARL outperforms commonly used base stock heuristic by 44.7% and the best performing RL method by up to 12.1% on average across different supply chain environments.

1 Introduction

Reinforcement learning (RL) has led to considerable breakthroughs in diverse areas such as games (Mnih et al. 2013), robotics (Kober, Bagnell, and Peters 2013) and others. Since RL provides a systematic framework to solve diverse problems with very limited domain knowledge, it has also been applied to other domains such as healthcare (Yu, Liu, and Nemati 2019). But the application of RL in real world problems poses unique challenges.

Many real world problems (e.g., inventory and revenue management), have large action spaces, specific state dependent action constraints, and underlying stochastic transition dynamics. For example, a retailer managing the inventory across a network of nodes in the supply chain has to decide how much inventory to place across the different nodes of the network. To accomplish this, the retailer has to account for (i) uncertain demand across the nodes in the network; (ii) a possible large set of feasible actions since the retailer decides on the number of units to allocate at different nodes; and (iii) a large number of constraints to ensure that the allocation remains feasible. These characteristics ensure that a direct application of existing RL methods remains limited (Gijsbrechts et al. 2018; Oroojlooyjadid et al. 2021; Sultana et al. 2020). Large action spaces render enumeration based techniques computationally infeasible. Hence, existing research has focused on either analyzing simplified inventory settings where a parameterized optimal policy can be constructed (Sun and Van Mieghem 2019; Xin 2021), or relevant constraints are relaxed and heuristics are used to estimate feasible solutions (Kunnunmak and Topaloglu 2008a; Federgruen, Guetta, and Iyengar 2018), or domain expertise is used to decide to approximate the state-space representation (Van Roy et al. 1997; Chen and Yang 2019).

The current paper takes a different approach to solving this problem. Our approach uses neural networks (NNs) to approximate the value function and uses ideas from mathematical programming (MP) and sample average approximation (SAA) to solve the per-step-action optimally. Our proposed framework is general and can be used to solve real world inventory management problems with complexities that make analytical solutions intractable (e.g. lost sales, dual sourcing with lead times, multi-echelon supply chains and many others).

We make the following contributions through this work:

1. We present a policy iteration algorithm for dynamic programming problems with large action spaces and underlying stochastic dynamics that we call Programmable Actor Reinforcement Learning (PARL). The algorithm uses a NN to approximate the value function along with techniques from SAA. In each iteration, the estimated NN is then used to generate an actor policy using integer-programming techniques.

2. To resolve the issue of computational complexity and underlying stochastic dynamics, we use techniques from SAA and discretization of continuous functions. Analytically, we show that for a given critic, the learned policy in each iteration converges to the optimal policy as the underlying samples of the uncertainty go to infinity. Practically, we show that if the underlying distribution of
the uncertainty is known, a properly selected discretization can yield near optimal actor policy even with very few samples.

3. We perform extensive computational experiments on real world inventory management settings and compare our proposed algorithm with state-of-the-art benchmark algorithms. We find that the proposed PARL algorithm is able to outperform both state-of-the-art machine learning (12.1% on average across different settings) as well as a standard inventory management heuristic (up to 44.7% on average across different settings). Our extensive simulation results provide a benchmark for various previously known intractable supply chain settings (network inventory management with lost sales, back order costs, stochastic demand and lead times), and could be of independent interest to researchers.

2 Literature Review

The current paper is related to three different areas:

Approximate Dynamic Programming (ADP): Our work is related to the broad field of ADP (Powell 2007). ADP methods use an approximation of the value function to optimize over computationally intractable dynamic programming problems. Traditionally, a set of features is chosen and polynomial functions of these features are used to approximate the value function. Naturally, one drawback that remains is that the quality of approximation depends on appropriately selecting the features as well as the functions for the approximation, which is not trivial. Hence, a NN can be used to approximate the value function, thereby replacing the step of feature and function selection (van Heeswijk and La Poutré 2019; Xu et al. 2020).

Mathematical programming (MP) based RL actor: MP techniques have recently been used for optimizing actions in RL settings with DNN-based function approximators and large action spaces. They leverage MP to optimize a mixed-integer (linear) problem (MIP) over a polyhedral action space using commercially available solvers such as CPLEX and Gurobi. A number of papers show how trained ReLU-based DNNs can be expressed as an MP with (Tjandraatmadja et al. 2020; Anderson et al. 2020) also providing ideal reformulations that improve computational efficiencies with a solver. (Ryu et al. 2019) propose a Q-learning framework to optimize over continuous action spaces using a combination of MP and a DNN actor. (Delarue, Anderson, and Tjandraatmadja 2020; van Heeswijk and La Poutré 2019; Xu et al. 2020) show how to use ReLU-based DNN value functions to optimize combinatorial problems (e.g., vehicle routing) where the immediate rewards are deterministic and the action space is vast. We extend such approaches and results to problems where the immediate reward can be uncertain as is the case with inventory management problems.

RL for inventory management: Early work that shows the benefits of RL for multi-echelon inventory management problems include (Van Roy et al. 1997; Giannoccaro and Pontrandolfo 2002; Stockheim, Schwind, and Koenig 2003). There has been a recent surge in using DNN-based reinforcement learning techniques to solve supply chain problems (Gijsbrechts et al. 2018; Oroojlooyjadid et al. 2021; Sultana et al. 2020) because the widely used base stock threshold policies are known to be optimal only in special cases (e.g., serial chain with back-ordered demands or the inability to hold demand in warehouses). See seminal works of (Clark and Scarf 1960; Federgruen and Zipkin 1984) and a recent review of multi-echelon inventory models studied in (de Kok et al. 2018). Optimal policy structures are unknown even in the single-node lost sales, dual sourcing settings and known heuristics are optimal in an asymptotic sense (see discussion and references in §1). A DNN-based actor critic method to solve the inventory management problem was studied in (Gijsbrechts et al. 2018) for the case of single node lost sales and dual sourcing settings, as well as multi-echelon settings, and showed improved performance in the latter setting. (Oroojlooyjadid et al. 2021) show how RL can be used to solve the classical bear game problem where agents in a serial supply chain compete for limited supply. More recently (Sultana et al. 2020) use a multi-agent A2C framework to solve an inventory management problem for a large number of products in a multi-echelon setting. Unlike these papers, we adopt an MP-based RL actor and show the benefit over vanilla DRL approaches. Our proposed method has the ability to factor in known state dependent constraints explicitly, as opposed to having to implicitly infer them.

3 PARL: Programming Actor Reinforcement Learning

We consider an infinite horizon discrete-time discounted Markov decision process (MDP) with the following representation: states \( s \in S \), actions \( a \in A(s) \), uncertain random variable \( D \in \mathbb{R}^{d \times m} \) with probability distribution \( P(D = d[s]) \) that depends on the context state \( s \), reward function \( R(s, a, D) \), distribution over initial states \( \beta \), discount factor \( \gamma \) and transition dynamics \( s' = T(s, a, d) \) where \( s' \) represents the next state. A stationary policy \( \pi \in \Pi \) is specified as a distribution \( \pi(s) \) over actions \( A(s) \) taken at state \( s \). Then the expected return of a policy \( \pi \in \Pi \) is given by \( J^\pi = \mathbb{E}_{s \sim \beta} V^\pi(s) \) where the value function is defined as \( V^\pi(s) = \sum_{t=0}^{\infty} \mathbb{E}[\gamma^t R(s_t, a_t, D_t) | s_0 = s, \pi, P, T] \). The optimal policy is given by \( \pi^* = \arg \max_{\pi \in \Pi} J^\pi \). The Bellman’s operator \( F[V](s) = \max_{a \in A(s)} \mathbb{E}_{D \sim P(\cdot/[s])} [R(s, a, D) + \gamma V(T(s, a, D))] \) over the state space is known to have a unique fixed point (i.e., to \( V = FV \)) at \( V^{\pi^*} \). This is crucial in the policy iteration scheme developed below that improves the learned value function and hence the policy over subsequent iterations.

We assume that the state space \( S \) is bounded, the action space \( A(s) \) is composed of discrete and/or continuous actions in a bounded polyhedron and lastly the transition dynamics \( T(s, a, d) \) and the reward function \( R(s, a, D) \) are piece-wise linear and continuous in \( a \in A(s) \).

We propose a Monte-Carlo simulation based policy-iteration framework where the learned policy is the outcome of a mathematical program which we refer to as PARL: Pro-
gramming Actor Reinforcement Learning (see Algorithm 1 and an illustrative block diagram in Appendix A.1). PARL is initialized with a random policy. The initial policy is iteratively improved over epochs with a learned critic (or the value function). In epoch $j$, policy $\pi_{j-1}$ is used to generate $N$ sample paths, each of length $T$. At every time step, a tuple of \{state, reward, next-state\} is also generated that is then used to estimate the value function $\hat{V}_\theta^{\pi_{j-1}}$ using a neural network parameterized by $\theta$. Particularly, in every epoch, for each sample path, we also get an estimate of the cumulative reward given by $Y_k(s_0^n) = \sum_{t=1}^T \gamma^{t-1} R_{kt}, \forall n = 1, \ldots, N$, where $s_0^n$ is the initial state of sample-path $n$. The initial states and cumulative rewards can be then passed on to a neural network which estimates the value of policy $\pi_{j-1}$ for any state, i.e., $\hat{V}_\theta^{\pi_{j-1}}$. Once a value estimate is generated, the new policy using the trained critic is simply

$$\pi_j(s) = \arg \max_{a \in A(s)} E_D \left[ R(s, a, D) + \gamma \hat{V}_{\theta}^{\pi_{j-1}}(T(s, a, D)) \right].$$

(1)

Problem (1) is hard to solve because of two main reasons. First, notice that $\hat{V}_{\theta}^{\pi_{j-1}}$ is a neural network which makes enumeration based techniques intractable, especially for settings where the actions space is large. And second, the objective function involves evaluating expectation over the distribution of uncertainty $D$ that is analytically intractable to compute. We next discuss how PARL addresses each of these complexities.

Optimizing over a neural network: Consider Problem (1) for a single realization of uncertainty $D$ given by $\max_{a \in A(s)} R(s, a, D) + \gamma \hat{V}_{\theta}^{\pi_{j-1}}(T(s, a, D))$. We describe a MP approach to solve this single-realization problem here and more generally next. We begin by assuming the value V-function is a trained $K$-layer feed forward ReLU-network with input state $s$ satisfies the following equations for $k = 2, \ldots, K$:

$$z_1 = s, \quad \tilde{z}_k = W_{k-1} \tilde{z}_{k-1} + b_{k-1}, \quad \hat{V}_\theta(s) := c^T \tilde{z}_K.$$ 

Here $\theta = (c, \{W_k, b_k\}_{k=2}^{K-1})$ are the weights of the V-network with $(W_k, b_k)$ being the multiplicative and bias weights of layer $k$ and $c$ being the weights of the output layer. Here $\tilde{z}_k, z_k$ denote the pre- and post-activation values at layer $k$. The non-linear equations re-written exactly as a MP with binary variables and $M$-constraints (Ryu et al., 2019, Anderson et al., 2020). For completeness, we briefly describe the steps.

Consider a neuron in the network with parameters $(w, b)$. For example, in layer $k$ neuron $i$'s parameters are $(W_k, b_k)$. Assuming a bounded input $x \in [l, u]$, the output $z$ of that neuron can be obtained with the following MP representation:

$$P(w, b, l, u) = \begin{cases} 
 z \geq w^T x + b, \\
 z \geq 0, \\
 z \leq w^T x + b - M^- (1 - y), \\
 z \leq M^+ y, \\
 x \in [l, u], y \in \{0, 1\}, z \in \mathbb{R}
\end{cases}$$

(2)

where $M^+ = \max_{x \in [l, u]} w^T x + b$ and $M^- = \min_{x \in [l, u]} w^T x + b$. Let $u_i = u_i$ if $w_i \geq 0$ and $l_i$ otherwise and, let $l_i \geq 0$ and $u_i$ otherwise. Hence $M^+ = w^T u + b$ and $M^- = w^T l + b$. Note that if $M^+ \leq 0$ (or if $M^- \geq 0$), the binary variable $y$ in MP can eliminated and the MP can be reduced to $z = 0$ (or $z = w^T x + b$ respectively).

Starting with the bounded input to the V-network, which can be derived from the bounded nature of $S$, the upper and lower bounds for subsequent layers can be obtained by assembling the $\max\{0, M^+\}$ and $\max\{0, M^-\}$ for each neuron from its prior layer. We will refer to them as $[l_k, u_k]$ for every layer $k$. This reformulation of the V-network combined with linear nature of the reward function $R(s, a, d)$ w.r.t $a$ and polyhedral description of the feasible set $A(s)$, lend themselves in reformulating Problem (1) as a MP for any given realization of $d$. In §5, we provide the corresponding formulation for the inventory management problem.

Maximizing expected reward with a large action space:

Problem (1) maximizes the expected profit where the expectation is taken over the uncertainty set $D$. Evaluating the expected value of the approximate reward is computationally hard. Hence, we take a SAA approach (Kim, Pasupathy, and Henderson 2015) to solve it. Let $d_1, d_2, \ldots, d_\eta$ denote $\eta$ independent realizations of the uncertainty $D$. Then, we let

$$\hat{\pi}_j(s) = \arg \max_{a \in A(s)} \frac{1}{\eta} \sum_{i=1}^\eta R(s, a, d_i) + \gamma \hat{V}_{\theta}^{\pi_{j-1}}(T(s, a, d_i)).$$

(3)

Problem (3) involves evaluating the objective only at sampled demand realizations. Assuming that for any $\eta$, the set of optimal actions is non empty, we show that as the number of samples, $\eta$ grows, the estimated optimal action converges to the optimal action. We make this statement precise in Proposition 1.

Proposition 1 Consider epoch $j$ of the PARL algorithm with a ReLU-network value function estimate $\hat{V}_{\theta}^{\pi_{j-1}}(s)$ for some fixed policy $\pi_{j-1}$. Suppose $\pi_j, \hat{\pi}_j$ are the optimal policies as described in Problem (1) and its corresponding SAA approximation respectively. Then, $\forall s,$

$$\lim_{\eta \to \infty} \hat{\pi}_j(s) = \pi_j(s).$$

Proposition 1 shows that the quality of the estimated policy improves as we increase the number of demand samples. Nevertheless, the computational complexity of the problem also increases linearly with the number of samples: for each demand sample, we represent the DNN based value function estimation using binary variables and the corresponding set of constraints.

We propose to use a weighting scheme in the special case when the uncertainty distribution $P(D = d(s)$ is known and independent across different dimensions. Let $d_1, d_2, \ldots, d_\eta$ denote $\eta$ quantiles (for example, evenly split between 0 to 1). Also let $F_j$ and $f_j$, $\forall j = 1, 2, \ldots, d_{\text{dim}}$, denote the cumulative distribution function and the probability density function of the uncertainty $D$ in each dimension respectively. Let $d_{ij} = w^T x + b$
$F^{-1}_j(q_i) \& w_{ij} = f_j(q_i), \forall i = 1, 2, ..., \eta, j = 1, 2, ..., \dim$ denote the uncertainty samples and their corresponding probability weights. Then, a single realization of the uncertainty is a $\dim$ dimensional vector $d_i = [d_{i1}, ..., d_{i,\dim}]$ with associated probability weight $w^\text{pool}_i = w_{i1} \ast w_{i2} \ast w_{i,\dim}$. With $\eta$ realizations of uncertainty in each dimension, in total there are $\eta^{\dim}$ such samples. Let $Q = \{d_i, w^\text{pool}_i\}$ be the set of demand realizations sub sampled from this set along with the weights (based on maximum weight or other rules) such that $|Q| = \eta$. Also let $w_i = \sum_{i\in Q} w^\text{pool}_i$. Then Problem (3) becomes

$$\hat{\pi}^\eta(s) = \arg \max_{a \in \mathcal{A}(s)} \sum_{d_i \in Q} w_i \left(R(s, a, d_i) + \gamma \hat{V}^\eta_{\hat{\theta}}(T(s, a, d_i))\right),$$

where $w_i = w^\text{pool}_i / w_Q$. The computational complexity of solving the above problem remains the same as before but since we use weighted samples, the approximation to the underlying expectation improves.

Algorithm 1 PARL

1: Initialize with random actor policy $\pi_0$. 
2: for $j \in [T]$ do 
3:  for (epoch) $n \in [N]$ do 
4:  Play policy $\pi_{j-1}$ for $T(1-\epsilon)$ and random action for $\epsilon T$ steps starting with state $s_{0}^\epsilon \sim \beta$. 
5:  Let $R_{\text{cum},n}^t \equiv \sum_{t'=1}^T R_{i}^t$ and store tuple $\{s_i^n, R_{\text{cum},n}^t\} \forall t = 1, ..., T$. 
6:  Approximate a DNN value function approximator by solving 
7:  $\hat{V}_j = \arg \min_{\hat{\theta}} \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^T (R_{\text{cum},n}^t - f(s_i^n, \hat{\theta}))^2$

8:  Sample $\eta$ realizations of the underlying uncertainty $D$ and obtain a new policy (as a lazy evaluation as needed) by solving either Problem (3) or (4) depending on the selected sampling method.

9: end for

4 Application of PARL to Multi-echelon Inventory Management

We now describe the application of PARL to the classic real-world multi-echelon inventory management problems in supply chain. We consider a firm managing inventory replenishment and distribution decisions for a single product across a network of stores (also referred to as nodes) with goal to maximize profits while meeting customer demands.

Let $\Lambda$ be the set of nodes, indexed by $l$. Each of the nodes can produce a stochastic amount of inventory in every period denoted by the random variable (r.v) $D_l^t$ which is either kept or distributed to other nodes. Any such distribution from node $i$ to $i'$ has a deterministic lead time $L_{ii'}^t \geq 0$ and is associated with a fixed cost $K_{ii'}^t$ and a variable cost $C_{ii'}^t$. Every node uses the inventory on-hand to fulfill local stochastic demand denoted by the r.v $D_l^t$ at a price $p_l$. We assume any excess demand is lost. If there is an external supplier, we denote it by a dummy node $S^\text{inf}$. For simplicity, we assume there is at most one external supplier and that the fill rate from that external supplier is 100% (i.e., everything that is ordered is supplied). We denote the upstream nodes that supply to node $l$ by the set $O_l \subset \Lambda \cup S^\text{inf}$. In every period the firm has to decide how much inventory to distribute from node to node and how much inventory should each node request from the external supplier. All replenishment decisions are have lower and upper capacity constraints denoted by $U^L_l$ and $U^U_l$. There is also holding capacity at every node denoted by $\hat{U}_l$. The firm’s objective is to maximize the overall profit. Assuming an i.i.d nature of stochasticity for each r.v, the firm’s problem can be modeled as an infinite horizon discrete-time MDP as follows:

$$V(I) = \max_{x_{il} \in Z^+} E_D \left[R(I, x, D) + \gamma V(I')\right]$$

where $R(I, x, D) = \sum_{l \in \Lambda} R_l(I_l, x_l, D_l)$, $\bar{R}_l(I_l, x_l, D_l) = p_l \min\{D^l_l, \tilde{\bar{R}}^l_l\} - \sum_{l' \in O_l} \left[K_{l'l}^l x_{l'l} > 0 + C_{l'l} x_{l'l}\right] - h_l I^l_l, \forall l \in \Lambda,$

$$I^l_l = I^l_l + \hat{I}^l_l + D^l + \sum_{l' \in O_l} x_{l'l} 1_{L_{ll'} = 0} -$$

$$\sum_{l' \in \Lambda \cup \{O_l\}} x_{ll'}, \forall l \in \Lambda,$$

$$I^0_l = \min \left\{U_l, \left[I^l_l - D^l_l\right]^+\right\}, \forall l \in \Lambda,$$

$$I^{l+1}_l = I^l_l + \sum_{l' \in O_l} x_{l'l} 1_{L_{ll'} = j}, \forall l \leq j \leq \max_{l' \in O_l} L_{ll'}, l \in \Lambda,$$

Here $I$ is the inventory pipeline vector for all nodes and the state space of the MDP, $x_l$ the action taken by the firm described by the vector of inventory movements from all other nodes to node $l$ at time $t$, $R_l(.)$ the reward function for each node $l$ described in Eq. (7). $I'$ the next state defined by the transition dynamics in Eqs. (9)–(10). These temporal inventory flow conservation equations ensure that, (i) when a step is taken, inventory in the pipeline for next period becomes inventory of the current period; (ii) all the current orders in the period are recorded in the inventory pipeline variable. These orders will arrive based on the lead time between the upstream and the downstream node. Similarly, auxiliary variables $\hat{I}^l_l$ defined in Eq. (6). The auxiliary variable has an interpretation of the total inventory in the system prior to meeting demand which stems from the on-hand inventory $I^l_l$, incoming pipeline inventory $I^l_l$, stochastic node production $D^l_l$, the incoming inventory from other nodes with lead time zero and the out-going inventory from this node.
Note that the state space $I$ is a collapsed state space compared to the inventory pipelines over connections between nodes as the reward $R_{pt}(\cdot)$ just depends on collapsed node inventory pipelines. Also, transportation cost and holding cost related to pipeline inventory are without loss of generality set to 0, as the variable purchase cost $C_{pt}$ can be modified according to account for these additional costs.

This setting models many real-world multi-echelon supply chain structures shown in Fig. [1]. The figures aim to show three types of nodes - supply nodes (S) that just produce inventory for downstream, warehouse nodes (W) that act as distributors and retail nodes (R) which face external demand. The supply node can be part of $\Lambda$ or be an external supplier $S_{\text{ext}}$. Example 1S-2W-3R (dual sourcing) depicts how sometimes nodes can have two inventory sources, commonly referred to in the supply chain literature as dual-sourcing setting.

![Example of different multi-echelon supply chain networks. In 1S-3R, a single supplier node serves a set of 3 retail nodes directly. In 1S-2W-3R, the supplier node serves the retail nodes through two warehouses. In 1S-2W-3R (dual sourcing), each retail nodes can is served by two distributors.](image)

It is easy to observe that the assumptions about PARL related to the state and action spaces, the reward and the transition dynamics are satisfied by the inventory management setting described here. In Appendix A.3, we provide the exact mixed-integer linear programming reformulation of the PARL actor for the inventory management MDP, using standard linearization techniques for the immediate reward and the $M$ reformulation for the value function part discussed in § [3]. In § [5] we provide computational results on the performance of PARL for various supply chain settings represented in Fig. [1].

As a note, in the MDP model, we assume excess demand is lost, while there can be settings such as in a B2B environment where the demands can be backordered. This extension is easy to include by allowing the current on-hand inventory to be negative (see Van Roy et al. [1997], Gijsbrechts et al. [2018] for a hybrid model). For a single node retail node $(|\Lambda| = 1)$ and one external supplier $S_{\text{ext}}$, the optimal policy for back-ordered demand has a $(s, S)$ structure where $S$ is referred to as the order-up-level based on the inventory position (sum of on-hand plus this in the pipeline) and $s$ an inventory position threshold, below which orders are placed (Clark and Scarf [1960]). This $(s, S)$ policy is commonly referred to as the base stock policy. In the lost sales setting that we consider, even with just a retail node, when lead times are non-zero, the structure of the optimal policy is unknown (Zipkin [2008a,b]) and (Huh et al. [2009], Goldberg et al. [2016]) prove structural results when $p/h \rightarrow \infty$. (Sheopuri, Janakiraman, and Seshadri [2010]) prove that the lost sales problem is a special case of dual sourcing problem (one retail node with 2 external suppliers), and thus, base stock policies are not optimal in general. Despite their non-optimality, they are popular both in practice and in the literature where authors restrict to the set of base stock policies for tractability reasons or to prove guarantees on the policy structure. For example, (Rong, Atan, and Snyder [2017]) propose heuristics for order-up to base stock levels in multi-echelon distribution (tree) networks without fixed ordering costs (also see Kunnumkal and Topaloglu [2008a,b]), Agrawal and Jin [2019] propose a learning-based method to find the best base stock policy in a single node lost sales setting with regret guarantees and Pirhooshyaran and Snyder [2020] develop a DNN-based learning approach to find the best order up-to levels in each link of a general supply chain network. Hence we benchmark PARL against base stock policies in the following section.

5 Computational Experiments

We develop a general purpose multi-echelon inventory management simulation environment defined with nodes (entities) and directional-connections (links). We model three types of entities - suppliers (S), warehouses (or distributors, W) and retailers (R). All entities are associated with holding costs, holding capacities and spillage costs, while retailers are additionally associated with price, demand uncertainties and a lost-sales demand type, and suppliers with production uncertainties. Each link is associated with order costs, lead time and maximum order quantity. The environment executes on the ordering and distribution actions specified by the agents by first ensuring its feasibility using a proportional fulfillment scheme (as it cannot send more than the inventory in a node) samples the uncertainties, accumulates the reward (the revenue from fulfillment less the cost of ordering and holding), and returns the next state to the agent.

We consider 7 different instances of this environment for our comparisons based on the 2 and 3 echelon structures described in Fig. [1]. We consider 4 variations of the 2 echelon supplier-retailer settings: 1S-3R-High, 1S-3R, 1S-10R, 1S-20R where high refers to higher production capacity compared to the 1S-3R system, and 3 variations of the 3 echelon system: 1S-2W-3R, 1S-2W-3R (DS), 1Sinf-2W-3R, with (DS) referring to dual sourcing and $S_{\text{ext}}$ referring to a supplier with infinite inventory (here downstream warehouses guaranteed to receive what is ordered in each time period). The specific details on the parameters for each of these environments are provided in Appendix A.5.

Benchmark Algorithms: We compare PARL with four state-of-the-art, widely used RL algorithms: PPO (Schulman et al. [2017]), TD3 (Fujimoto, Hoof, and Meger [2018]), SAC (Haarnoja et al. [2018]), and A2C (Mnih et al. [2016]); a popularly used $(s, S)$ base stock (BS) policy (Snyder and Shen [2019]) for each link; and a decomposition-aggregation (DA) heuristic (Rong, Atan, and Snyder [2017]) to evaluate order-up to policies $(s, S)$ in near closed form.

For the RL algorithms we used the tested and reliable implementations provided by Stable-Baselines3 (Kafli et al. [2019], under the MIT License. We made all our environment
compatible with OpenAI Gym \cite{Brockman2016} and to implement PARL we built on reference implementations of PPO provided in SpinningUp \cite{Achiam2018} (both MIT License). We ran RL baselines on a 152 node X 26 (average) CPU cluster (individual jobs used 1 CPU and max <1GB RAM), and PARL on a 13 nodes X 48 (average) CPU cluster (individual PARL job uses 16 CPUs for trajectory parallelization and CPLEX computations and average <4GB RAM). We use version 12.10 of CPLEX with a time constraint of 60s per decision step with 2 threads.

Parametric \((s, S)\) base stock policies are discussed for retail nodes with infinite capacity upstream supplier \cite{Synder2019}. In this policy, if \(I\) is the inventory pipeline vector for a retailer, the inventory position is defined as \(IP = \sum_{i=0}^{L} P\), where \(L\) is the lead time from the supplier, and the order quantity is \(\text{max}\{0, S - IP\}\) as long as \(IP <= s\) and 0 otherwise. We use this idea to implement a heuristic base stock policy as follows. For the 2-echelon \(1S - nR\) environments, we identify the best base stock policy via grid search for each link using a \(1S - 1R\) environment. Recall that if the retailers order to order, they receive inventory proportional to the request due to the proportional fulfillment strategy. For this policy structure, many papers \cite{Axsater1993} have shown the optimality of such a decomposition in 2-echelon distribution networks. As the proposed policies in these papers are derived under no-fixed cost and backorder assumptions, instead of using their proposed policies, we perform a grid search, which even though is computationally expensive, returns the best policy under the assumed policy structure. For the 3-echelon \(1S - nR\) environments, we use the same strategy for the \(W - R\) links but computing inventory positions \(IP\) based on the lead time for that link (note that inventory pipelines can be longer than lead-time in the dual sourcing setting). For the \(S - W\) links we use environments that treat the warehouse as a retailer with demand equal to the sum of the downstream (lead time) retail nodes with infinite capacity upstream supplier \cite{Synder2019}.

Parameter tuning and Evaluation: We perform extensive tuning of different hyper parameters (HPs) of the benchmark RL algorithms. We first evaluated performance across environments for a large random grid of HP combinations to narrow down the set of candidate HP values to a reasonable subset. This resulted in: a set of candidate gamma values; fixing observation and action space representations to be continuous (discrete and multi-discrete representations consistently performed worse, likely because of their larger dimension); setting activations to ReLU (consistently worked comparable or better than TanH); fixing the network architectures to the standard 64x64 as we did not see benefits from larger or different architectures; fixing epoch length where applicable to 2048 steps (worked better than shorter in initial PPO experiments); fixing a set of learning rates and value function coefficients to try; and fixing the batch size to the standard 64. In the end we defined a grid of 32 to 36 HP combinations for each benchmark method (varying gamma, learning rates, and exploration options) and ran 10 different randomly seeded modeling runs for each combination per method and environment. We then computed the average accuracy per epoch (using 20 evaluation episodes) across the 10 runs for each method, environment and HP combo. We then selected the HP combo for each method and environment that gave the maximum mean reward as its best HP combo.

For the PARL algorithm, because of computational constraints, we only tune two parameters: learning rate and number of samples to be used for solving the SAA problem per time step, 3 values each. For base stock, the main HP is the granularity of the grid search, which was set to 2 units. DA has no HPs.

Then for all methods, given a selected best hyper-parameter combination per method and environment, to perform the evaluation we then ran 10 different training runs for each (i.e., with different random seeds). Finally we took the best epoch model according to evaluation scoring from each of those 10 runs as the best model per run, and evaluated each of the 10 with 20 episodes to get our final reported mean and standard deviation per method and environment. Additional and complete details of the hyper parameter tuning procedure including final range used and selected hyper-parameters are provided in Appendix A.4.

Performance: In Table \ref{table:performance} we present the average per step reward (over test runs) of the different algorithms and compare them to PARL in 7 different settings described earlier. We additional provide the percentage improvement over two widely used methods: PPO and BS. We observe that PARL is a top performing method, in fact it outperforms all benchmark algorithms in all but one setting, 1S\textsuperscript{1st}-2W-3R. On average across the different supply chain settings, PARL outperforms the best performing RL algorithm by 12.1% and the BS policy by 44.7%.

Notably, the improvements are higher in supply chain settings that are more complex (1S-20R, 1S-10R, 1S-2W-3R and 1S-2W-3R (DS)) amongst the settings tested in the paper. While in the 10R and 20R settings, the retailer has to optimize decisions over a larger network with larger action space, 1S-2W-3R and 1S-2W-3R (DS) are multi-echelon settings with more complex supply chain structure. Similarly, in the 1S-3R setting, the supplier is more constrained than the 1S-3R-High setting, which makes the inventory allocation decision more complex. PARL’s ability to explicitly factor in known state-dependent constraints enables it to out-perform other methods in these settings. Settings 1S-3R-High and 1S\textsuperscript{1st}-2W-3R have relatively high supplier production, where BS and related heuristics like DA are known to work well. Here PARL is on par with the BS heuristic (within one standard deviation of the BS heuristic’s performance) and out-performs other RL methods. In these settings, the combination of surplus inventory availability, coupled with low holding cost and high demand uncertainty, encourage holding high inventory levels over long time-horizons. This makes reward attribution for any specific ordering action harder, and thus these settings are harder to learn for RL algorithms.

We also analyze the rate of learning of different algo-
algorithms during training. In Figure 2, we plot the average per-step reward over training steps from 3 different environments. We find that in both cases, the PARL actor performs much worse in the initial training runs on account of optimizing over a poorly trained critic. Once, the critic improves in accuracy, PARL is able to recover a very good policy during training.

Finally, we report algorithm run-times. The average per-step run time of the PARL algorithm is 0.178, 0.051, 0.050, 0.089, 0.051, 0.044 and 0.042 seconds in the 1S-3R-High, 1S-3R, 1S-10R, 1S-2W-3R, 1S-2W-3R (DS) and 1S-2W-3R settings, respectively. The average per step run time of PPO (the DRL algorithm that performs the best in most settings) is 0.007, 0.005, 0.010, 0.008, 0.008, 0.008, and 0.007 seconds respectively. Clearly, PPO outperforms PARL in terms of run-time. This is because while per-step action in PARL is an outcome of an integer-program, DRL algorithms take gradient steps that are computationally much faster.

Table 1: Average per-step-reward with standard deviation and median (within brackets) of different benchmark algorithms, averaged over different testing runs. We bold all top performing methods: those with performance not statistically significantly worse than the best method, using one standard deviation.

| Setting       | SAC       | TD3       | PPO       | A2C       | BS        | DA        | PARL       | PARL over PPO / BS |
|---------------|-----------|-----------|-----------|-----------|-----------|-----------|------------|-------------------|
| 1S-3-R-High   | 478.8 ± 8.5 | 374.7 ± 15.7 | 499.4 ± 5.7 | 490.8 ± 8.9 | 513.3 ± 5.9 | 513.0       | 477.4 ± 4.6 | **514.8 ± 5.3** / 3.1% / 0.3% |
| 1S-3-R        | 398.0 ± 3.2 | 329 ± 45.2 | 397.0 ± 1.6 | 392.4 ± 4.4 | 392.87     | 313.7 ± 3.1 | 304.0 ± 2.0 | **400.3 ± 3.3** / 0.8% / 27.6% |
| 1S-10R        | 870.5 ± 68.9 | 744.4 ± 71.4 | 918.3 ± 24.7 | 768.1 ± 40.5 | 660.5 ± 2.1 | 653.4 ± 1.0 | 659.9       | 6006.3 ± 29.5 / 9.6% / 52.3% |
| 1S-20R        | 1216.1 ± 25.2 | 1105.8     | 1117.3 ± 37.5 | 1114.4     | 862.7 ± 3.0 | 851.6 ± 1.2 | 851.9       | **1379.2 ± 190.1** / 28.5% / 59.9% |
| 1S-2W-3R      | 374.2 ± 3.7 | 361.1 ± 15.4 | 360.2 ± 23.2 | 302.2       | 302.2 ± 0.5 | 276 ± 2.1 | 276.6       | 398.3 ± 2.5 / 5.5% / 32.4% |
| 1S-2W-3R (DS) | 344.1 ± 20.6 | 259.3 ± 32.3 | 327.5 ± 32.7 | 166.2 ± 3.8 | 157.6 ± 1.0 | 157.4       | **405.4 ± 2.0** / 4.5% / 123.9% |
| 1Sinf-2W-3R   | 40.6 ± 59.8 | 210.0 ± 57.0 | 136.8 ± 18.4 | 62.9 ± 33.7 | **208.8 ± 5.2** | 157.4 ± 3.8 | 201.5 ± 16.9 | 201.8 ± 16.9 / 47.3% / -3.5% |

Figure 2: Learning curves of PARL and benchmark algorithms during training runs.

6 Conclusions and Discussion

We develop a novel RL algorithm to solve the problem of inventory management over complex networks. Our proposed solution combines ideas from SAA, MP and traditional RL techniques and we show that the method outperforms state-of-the-art RL as well as inventory management methods in various supply chain settings. Through extensive computations, we also provide the first benchmark results for various RL algorithms on diverse supply chain settings.

This work also opens up various directions of future research. While the current work used parallelization to improve computational speed of PARL, further improvements in run time can be made from developing GPU based LP/IP solvers to increase scalability. This can also be achieved by using sparse NNs for value function approximation, or combining the MP based actor with parametric policies.

Another direction of future research is to increase robustness of PARL to changing critic. Since PARL takes deterministic actions that optimize over the learned critic, the method’s performance can be affected in cases when the critic provides a poor approximation of the value function. This can be improved by using techniques from robust optimization to optimize actions over uncertain NN parameters. Finally, developing more informed sampling techniques to improve expected value approximation with very limited
demand samples also remains an interesting direction that could lead to substantial improvements in run time without affecting the overall performance of the learned policy.

References

Achiam, J. 2018. Spinning Up in Deep Reinforcement Learning, https://github.com/openai/spinningup.

Agrawal, S.; and Jia, R. 2019. Learning in structured mdps with convex cost functions: Improved regret bounds for inventory management. In Proceedings of the 2019 ACM Conference on Economics and Computation, 743–744.

Anderson, R.; Huchette, J.; Ma, W.; Tjandraatmadja, C.; and Vielma, J. P. 2020. Strong mixed-integer programming formulations for trained neural networks. Mathematical Programming, 1–37.

Axsäter, S. 1993. Optimization of order-up-to-s policies in two-echelon inventory systems with periodic review. Naval Research Logistics (NRL), 40(2): 245–253.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI Gym. arXiv:1606.01540.

Chen, W.; and Yang, H. 2019. A heuristic based on quadratic approximation for dual sourcing problem with general lead times and supply capacity uncertainty. ISE Transactions, 51(9): 943–956.

Clark, A. J.; and Scarf, H. 1960. Optimal policies for a multi-echelon inventory problem. Management science, 6(4): 475–490.

de Kok, T.; Grob, C.; Laumanns, M.; Minner, S.; Rambau, J.; and Schade, K. 2018. A typology and literature review on stochastic multi-echelon inventory models. European Journal of Operational Research, 269(3): 955–983.

Delarue, A.; Anderson, R.; and Tjandraatmadja, C. 2020. Reinforcement Learning with Combinatorial Actions: An Application to Vehicle Routing. Advances in Neural Information Processing Systems, 33.

Federgruen, A.; Guetta, C. D.; and Iyengar, G. 2018. Two-echelon distribution systems with random demands and storage constraints. Naval Research Logistics (NRL), 65(8): 594–618.

Federgruen, A.; and Zipkin, P. 1984. Approximations of dynamic, multilocation production and inventory problems. Management Science, 30(1): 69–84.

Fujimoto, S.; Hoof, H.; and Meger, D. 2018. Addressing Function Approximation Error in Actor-Critic Methods. In International Conference on Machine Learning, 1582–1591.

Giannoccaro, I.; and Pontrandolfo, P. 2002. Inventory management in supply chains: a reinforcement learning approach. International Journal of Production Economics, 78(2): 153–161.

Gijsbrechts, J.; Boute, R. N.; Van Mieghem, J. A.; and Zhang, D. 2018. Can deep reinforcement learning improve inventory management? performance and implementation of dual sourcing-mode problems. Performance on Dual Sourcing, Lost Sales and Multi-Echelon Problems.

Glorot, X.; Bordes, A.; and Bengio, Y. 2011. Deep sparse rectifier neural networks. In Proceedings of the fourteenth international conference on artificial intelligence and statistics, 315–323. JMLR Workshop and Conference Proceedings.

Goldberg, D. A.; Katz-Rogozhnikov, D. A.; Lu, Y.; Sharma, M.; and Squillante, M. S. 2016. Asymptotic optimality of constant-order policies for lost sales inventory models with large lead times. Mathematics of Operations Research, 41(3): 898–913.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In Dy, J.; and Krause, A., eds., Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research, 1861–1870. PMLR.

Hara, K.; Saito, D.; and Shouno, H. 2015. Analysis of function of rectified linear unit used in deep learning. In 2015 International Joint Conference on Neural Networks (IJCNN), 1–8.

Huh, W. T.; Janakiraman, G.; Muckstadt, J. A.; and Rusmevichientong, P. 2009. Asymptotic optimality of order-up-to policies in lost sales inventory systems. Management Science, 55(3): 404–420.

Kim, S.; Pasupathy, R.; and Henderson, S. G. 2015. A guide to sample average approximation. Handbook of simulation optimization, 207–243.

Kober, J.; Bagnell, J. A.; and Peters, J. 2013. Reinforcement learning in robotics: A survey. The International Journal of Robotics Research, 32(11): 1238–1274.

Kunnumkal, S.; and Topaloglu, H. 2008a. A duality-based relaxation and decomposition approach for inventory distribution systems. Naval Research Logistics (NRL), 55(7): 612–631.

Kunnumkal, S.; and Topaloglu, H. 2008b. Using stochastic approximation methods to compute optimal base-stock levels in inventory control problems. Operations Research, 56(3): 646–664.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In International conference on machine learning, 1928–1937. PMLR.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

Oroojlooyjadid, A.; Nazari, M.; Snyder, L. V.; and Takač, M. 2021. A Deep Q-Network for the Beer Game: Deep Reinforcement Learning for Inventory Optimization. Manufacturing & Service Operations Management.

Pirhooshyan, M.; and Snyder, L. V. 2020. Simultaneous Decision Making for Stochastic Multi-echelon Inventory Optimization with Deep Neural Networks as Decision Makers. arXiv preprint arXiv:2006.05608.
Powell, W. B. 2007. Approximate Dynamic Programming: Solving the curses of dimensionality, volume 703. John Wiley & Sons.

Raffin, A.; Hill, A.; Ernestus, M.; Gleave, A.; Kanervisto, A.; and Dormann, N. 2019. Stable Baselines3. [https://github.com/DLR-RM/stable-baselines3].

Rong, Y.; Atan, Z.; and Snyder, L. V. 2017. Heuristics for base-stock levels in multi-echelon distribution networks. Production and Operations Management, 26(9): 1760–1777.

Ryu, M.; Chow, Y.; Anderson, R.; Tjandraatmadja, C.; and Boutilier, C. 2019. CAQL: Continuous action Q-learning. arXiv preprint arXiv:1909.12397.

Shapiro, A. 2003. Monte Carlo sampling methods. Handbooks in operations research and management science, 10: 353–425.

Shapiro, A.; Dentcheva, D.; and Ruszczyński, A. 2014. Lectures on stochastic programming: modeling and theory. SIAM.

Sheopuri, A.; Janakiraman, G.; and Seshadri, S. 2010. New policies for the stochastic inventory control problem with two supply sources. Operations research, 58(3): 734–745.

Snyder, L. V.; and Shen, Z.-J. M. 2019. Fundamentals of supply chain theory. John Wiley & Sons.

Stockheim, T.; Schwind, M.; and Koenig, W. 2003. A reinforcement learning approach for supply chain management. In 1st European Workshop on Multi-Agent Systems, Oxford, UK.

Sultana, N. N.; Meisher, H.; Baniwal, V.; Nath, S.; Ravindran, B.; and Khadilkar, H. 2020. Reinforcement Learning for Multi-Product Multi-Node Inventory Management in Supply Chains. arXiv preprint arXiv:2006.04037.

Sun, J.; and Van Mieghem, J. A. 2019. Robust dual sourcing inventory management: Optimality of capped dual index policies and smoothing. Manufacturing & Service Operations Management, 21(4): 912–931.

Tjandraatmadja, C.; Anderson, R.; Huchette, J.; Ma, W.; Patel, K.; and Vielma, J. P. 2020. The convex relaxation barrier, revisited: Tightened single-neuron relaxations for neural network verification. arXiv preprint arXiv:2006.14076.

Xin, L. 2021. Understanding the performance of capped base-stock policies in lost-sales inventory models. Operations Research, 69(1): 61–70.
Proof of Proposition 1

Proposition 2 Consider epoch \( j \) of the PARL algorithm with a ReLU-network value-to-go estimate \( \hat{V}_{\theta}^{\pi_{j-1}}(s) \) for some fixed policy \( \pi_{j-1} \). Suppose \( \pi_{j}, \hat{\pi}_{\eta}^{j} \) are the optimal policies as described in Problem (1) and its corresponding SAA approximation respectively. Then, \( \forall s, \lim_{\eta \to \infty} \hat{\pi}_{\eta}^{j}(s) = \pi_{j}(s) \), where \( \pi_{j}(s) \) is as described in Problem (1) and \( \hat{\pi}_{\eta}^{j}(s) \) is the corresponding SAA approximation.

Proof Consider any state \( s \) and let \( g(s, a, d) = R(s, a, d) + \gamma \hat{V}_{\theta}^{\pi_{j-1}}(T(s, a, d)) \). We start by showing that \( g^{\eta}(s, a, d) \) uniformly converges to \( \mathbb{E}[g(s, a, D)] \) with probability 1. We prove this result by proving two main properties of \( g(s, a, d) \): (i) \( g(s, a, d) \) is continuous in \( a \) for almost every \( d \in D \), and (ii) \( g(s, a, d) \) is dominated by an integrable function. To prove (ii), we show that \( g(s, a, d) \leq C < \infty \) w.p. 1 \( \forall a \in A(s) \).

First, notice that \( g(s, a, d) \) is an affine function of the immediate reward \( R(s, a, d) \) and NN approximation of the value-to-go function. By assumption, the immediate reward follows these properties. Hence, to show these properties for \( g(s, a, d) \), we only need to illustrate that the value-to-go estimation also follows these properties.

Consider the value-to-go approximation, simply denoted as \( \hat{V}_{\theta}(T(s, a, d)) \) with \( \theta = (c, \{(W_k, b_k)\}_{k=1}^{K-1}) \) denoting the parameters of the \( K \)-layer ReLU-network. As \( T(s, a, d) \) is continuous and \( \hat{V}_{\theta}(s) \) is continuous, \( \hat{V}_{\theta}(T(s, a, d)) \) is continuous. Note that \( T(s, a, d) \) lies in a bounded space for any realization of the uncertainty \( d \). Furthermore, since the parameters of the NN \( \theta \) are bounded, the outcome of each hidden layer, and subsequently the outcome of the NN are also bounded. This proves that the NN is uniformly dominated by an integrable function. Then, following Proposition 8 of (Shapiro 2003), we have uniform convergence of \( g^{\eta}(s, a, d) \) to \( \mathbb{E}[g(s, a, D)] \) w.p. 1. Finally, convergence of the optimal solution follows from a direct application of Theorem 5.3 of (Shapiro, Dentcheva, and Ruszczyński 2014), where we have used the fact that for all \( s \) the set of feasible actions is a bounded polyhedron \( A(s) \) and that for any \( \eta \), the set of optimal actions \( \hat{\pi}_{\eta}(s) \) is non-empty. This proves the final result.

While the above proof assumes that the action space is continuous, one can extend the results in the case of discrete action spaces as well. See (Kim, Pasupathy, and Henderson 2015) for a discussion on the techniques used for extending the analysis to this setting.

Math-Programming Actor in PARL for Inventory Management

Below we show mixed-integer linear reformulation of the inventory management MDP described in §4 using PARL for the value-to-go terms. This formulation can be solved using
commercially available standard optimization software such as CPLEX and Gurobi.

\[ V(I) = \max_{x_i \in Z^+} \sum_{i=1}^{n} w_i [R(I, x, d_i) + \gamma c^T z_{K_i}] \]

where

\[ R(I, x, d_i) = \sum_{l \in \Lambda} R_l(I_l, x_l, d_{li}) \quad \forall i, \]

\[ R_l(I_l, x_l, d_{li}) = p_l s_{li} - \sum_{l' \in O_l} [K_{l'l} w_{l'l} + C_{l'l} x_{l'l}] - h_l I_{li}^0 - \delta B_{li}, \quad \forall l \in \Lambda, i \]

\[ s_{li} \leq d_{li}^d, \quad \forall l \in \Lambda, i, \]

\[ s_{li} \leq I_{li}^0, \quad \forall l \in \Lambda, i, \]

\[ w_{l'l} \leq x_{l'l}, \quad \forall l \in O_{l'}, l' \in \Lambda, \]

\[ x_{l'l} \leq U_{l'l}^M w_{l'l}, \quad \forall l \in O_{l'}, l' \in \Lambda, \]

\[ \bar{I}_{li}^0 = I_{li}^0 + I_{li}^1 + d_{li}^p + \sum_{l' \in O_l} x_{l'l} 1_{l_{l'l}=0} - \sum_{l' \in O_l} x_{l'l}, \quad \forall l \in \Lambda, i, \]

\[ I_{li}^0 = I_{li}^0 - s_{li} - B_{li}, \quad \forall l \in \Lambda, i, \]

\[ I_{li}^0 = I_{li}^0 + \sum_{l' \in O_l} x_{l'l} 1_{l_{l'l}=j} - B_{li}, \quad \forall 1 \leq j \leq \max_{l' \in O_l} L_{l'l}, l \in \Lambda, i, \]

\[ B_{li} \geq 0, \quad \forall j = 0, \ldots, \max_{l' \in O_l} L_{l'l}, l \in \Lambda, i, \]

\[ (I_{l+1}^q, z_{k+1}, y_{k+1}) \in P(W_{k}, b_k, l_k, u_k) \quad \forall q \in N_k, \]

\[ (z_{k, i}, z_{k+1}, y_{k+1}) \in P(W_{k}, b_k, l_k, u_k) \quad \forall q \in N_k, k = 2, \ldots, K - 1. \]

We describe the extra notation that use in formulating the PARL actor as a MIP. Let \( I_{l+1}^q \) be the vector of \( I_{l+1}^l \) across all the locations \( l \) which is an input to the DNN for every uncertainty sample \( i \). We denote \( N_k \) as the number of neurons indexed by in layer \( k \) of the ReLU-network and it is indexed by \( q \). We let \( l_k \) and \( u_k \) are pre-computed lower and upper bounds of the inputs to every layer of the NN given the fixed bounds \([0, \bar{U}]\) of the first later and computed as described in \( \delta[3] \). We introduce a binary variable \( w_{l'l} \), which is 1 if \( x_{l'l} > 0 \) and 0 otherwise. This is enforced with Eqs. (15). A sales variable \( s_{li} \) is modeled via Eqs. (14). \( d_{li}^p \) will be exactly the minimum of the demand and inventory. \( B_{li} \) is decision variable that captures the inventory spilled over which is positive if the state update variables \( \bar{I}_{li} \) exceeds \( I_{li} \) and 0 otherwise. This condition is enforced using a small linear penalty term \( \delta B_{li} \) in the objective.

**Hyperparameters and parameter tuning**

In this section, we discuss the different parameters selected for PARL and the other benchmark methods and the corresponding tuning procedure.

**Fixed hyper parameters** Here we report the fixed set of hyper parameters used by all methods. These were determined based on two factors: (1) the commonly used settings across the RL literature (for example 64x64 architecture and batch size 64 is most commonly used across many different problems and methods), and by sampling random combinations from a large grid of hyper parameters and comparing results trends to narrow down the set of hyper parameters to consider to consistently well-performing values and reasonable ranges.

This was an iterative process where we tried a range of hyper parameters, then refined. We focused mostly on the PPO method at first as it was the first one we had implemented and started testing in this supply chain setting, but it gave us a general sense of what kind of hyper parameters had a chance at working well for these problems and environments. In particular, we tried larger network architectures, including 128x128, 512x512, 1024x1024, 128x128x128, 512x256, 1024x512, 1024x128, 128x32, 512x128, 512x256x64, but generally did not see significant improvement across environments, especially when using the continuous action and state spaces (perhaps also because the limited size of our observation and action spaces) - so decided to fix everything to the standard 64x64 for fair comparison, and improved computationally efficiency. We tried 32, 64, and 128 batch size, but also did not see significant difference in what gave the best results, so set this to the most commonly used 64. We also consistently saw ReLU activation performing as good or better than tanh (overall it gave close but slightly better results) - so fixed the activation to ReLU across methods for fair comparison, and since ReLU is known to enable more efficient optimization
Tuning hyper parameters  Here we show the selected set of best hyper parameters for each benchmark RL method and environment, in Table 6. These were selected based on what gave the best average reward (maximum over the training epochs), averaged across 10 different model runs for that hyper-parameter combination.

For PARL, a common set of hyper-parameters were used across all settings, with the exception of the discount factor, gamma, for the infinite supplier inventory setting, since we observed universally higher gamma being necessary for the baseline DRL methods. The discount factor was set to 0.75 (0.99 for the infinite supplier inventory setting only), learning-rate was set to 0.001, and the sample-averaging approach used was quantile sampling with 3 demand-samples per step.

Note that higher discount factor, gamma, was universally selected for the infinite supplier inventory environment, providing evidence for our previously mentioned hypothesis (in the main paper) that in this setting it was important to consider the impact of decisions over longer time/step horizons.

Supply chain environment problem parameters

In the table below we provide the details of the environment parameters in a concise format for the 7 different supply chain networks that we study and we describe it below.

We assume deterministic production constant per-period production and that the variability is only in the demand. The parameters are provided node type - Retailer (R), Supplier (S or S^{inf}) and Warehouse (W) - and then by links between them. Whenever they are provided in a list format, they correspond to the retailers and warehouses in a chronological order (i.e., R1, R2, R3 or W1, W2). Also, when there are more nodes or links than the parameters (few elements in the list specified in the table), it means the parameters list in repeated in a cyclic fashion. For example the lead time (S or W to R) for the environment 1S-10R is given by [1,2,3] and this means the lead time for links [S-R1, S-R2,...,S-R10] is (1,2,3,1,2,3,1,2,3,1). The notation for the distributions used are \( N(\mu, \sigma) \) for a normal distribution with mean \( \mu \) and standard deviation \( \sigma \) and \( U(a, b) \) discrete uniform between \( a \) and \( b \). Note that because demand is discrete and positive, when we use a normal distribution, we round and take the positive part of the realizations. The lead-time list has a tuple representation in the dual-sourcing setting to represent the lead time of a retailer from the two different warehouses. For example (1,5) in the list represents the lead time for W1-R1 and W2-R2. Lastly, the environment only imposes spillage cost at the node if the on-hand exceeds the holding capacity, while PARL MIP actor imposes it on the pipeline inventory actor to ensure it is not over-ordering. As the maximum order is less or equal to the holding capacity on various links,

--

1https://stable-baselines3.readthedocs.io/en/master/
such a constraint in PARL actor helps to avoid over-ordering in dual sourcing settings. The ability of PARL to gainfully incorporate such constraints exactly when they are known, possibly provides it an edge over vanilla RL methods.

Then for all methods, given a selected best hyper-parameter combination per method and environment, we trained 10 different models for each (i.e., with different random seeds). Each training run used an episode (trajectory) length of 256, an epoch length of 2048 and 500 epochs. Finally, for each trained model, we took the best epoch according to the observed reward during training, and then evaluated each of the 10 trained models with 20 test episodes, initialized with random seeds, and using trajectory length 256, to get our final reported mean and standard deviation per method and environment.

The resultant number of variables and constraints in the PARL MIP depend on the size of the NN used for approximating the value function (described in Table 3), and the topology of the supply-chain. The range of sizes for environments discussed in Table 1 are 1265-1877 variables (with up to 404 binary in the largest environment, the 1S-20R) and 2040-2686 constraints.

**Comparison of quantile and random sampling in PARL**

Here we compare the use of quantile sampling and random sampling to generate realizations of the uncertainty in $h_r$. For the five settings under consideration, we compare the per-step reward and per-step training time across the two sampling approaches.

As can be seen in Table 8, random sampling yields per-step rewards which are close to those obtained via quantile sampling. In terms of training time, random sampling is slower in certain settings (e.g. 1S-3R-High and 1S-10R), with higher per-step train-time average and variance.

**Decomposition-Aggregation (DA) Heuristic (Rong, Atan, and Snyder 2017)**

In this section, we describe our implementation of the DA heuristic. Note that we adaptation of the DA heuristic to the case of a Normal demand distribution as the authors discuss the method in the case of a Poisson demand distribution.

For 1S-nR environments, for every retailer $r$ compute its respective order up to level $S_r = F_{D_r}^{-1} \left( \frac{b_r}{b_r + h_r} \right)$ where $F_{D_r}^{-1}$ is the inverse cumulative demand distribution of the random variable $D_r \sim N(\mu_r, \sigma_r, \sqrt{L_r})$. Here $b_r$ is the retailer revenue per item less the variable ordering cost from the supplier and $h_r$ is the holding cost at the retailer. In every period, the retailer orders $S_r - IP_r$, where $IP_r$ is the retailer’s inventory position which is the sum of the retailer’s on-hand and that in the pipeline vector, in all a total of $L$ terms including on-hand.

For the 1S-2W-nR environments, we first decompose by sample paths $1S - 1W - 1R$.

For each such sample path, we compute $S_r = F_{D_r}^{-1} \left( \frac{b_r + h_{w_r}}{b_r + h_r} \right)$ where $F_{D_r}^{-1}$ is the inverse cdf of the random variable $D_r \sim N(\mu_r, \sigma_r, \sqrt{L_r})$. Here $b_r$ is the retailer revenue per item less the variable ordering cost from the supplier, $h_r$ is the holding cost at the retailer and $h_{w_r}$ is the holding cost of the warehouse in the sample path of interest.

We then compute $q_{w_r} = \ldots$
Table 4: Tuning hyper parameters and additional fixed hyper parameters for SAC and TD3 - we vary gamma, learning rate, and exploration options - resulting in 32 hyper parameter combinations

| Hyper Parameters for SAC and TD3                      | Value(s)                      |
|-------------------------------------------------------|-------------------------------|
| Discount Factor - Gamma (G)                           | 0.99, 0.9, 0.80, 0.75         |
| Learning rate (LR)                                    | 0.01, 0.003, 0.0003, 0.00003  |
| Use generalized State Dependent Exploration vs. Action Noise Exploration (SAC) or Action Noise vs. not (TD3) (EO) | True, False                   |
| Tau (soft update coefficient)                         | 0.005                         |
| Replay buffer size                                    | 10^5                          |
| Entropy regularization coefficient (SAC only)          | auto                          |

Table 5: Tuning hyper parameters for PARL

| Hyper Parameters for PARL                      | Value(s)                      |
|-----------------------------------------------|-------------------------------|
| Discount Factor - Gamma (G)                   | 0.99, 0.9, 0.8, 0.75          |
| Learning rate (LR)                            | 0.01, 0.003, 0.001            |
| Sample approximation averaging (SAA) approach used to generate demand samples | quantile, random              |
| SAA samples per step                          | 2, 3                          |

F \left[ 0.5F^{-1} \left[ \frac{b_r}{L_r} + h_r \right] + 0.5F^{-1} \left[ \frac{b_r}{L_r + h_w} \right] \right] \quad \text{where} \quad F \quad \text{and} \quad F^{-1} \quad \text{refers to the cdf and inverse cdf of the standard normal distribution N(0,1). We use this to compute echelon order up to level of the warehouse} \quad S_{w_r} = F_{D_{w_r}}^{-1} \left[ q_{w_r} \right] \quad \text{where} \quad D_{w_r} \quad \text{is distributed as} \quad N(\mu_r(L_r + L_w), \sigma_r \sqrt{L_r + L_w}).

An expected shortfall is computed which is \quad Q_{D_{w_r}}(s_{w_r}) = E_{D_{w_r}}[D_{w_r} - s_{w_r}]^+ \quad \text{where} \quad s_{w_r} = S_{w_r} - S_r.

Next we aggregate across sample paths to recompute the order up to level at common warehouse \textit{w} using a back-order matching method described as follows: \quad S_w = Q_{D_w}^{-1} \left( \sum_{r|w_r = w} Q_{D_{w,r}}(s_{w_r}) \right) \quad \text{where} \quad D_w \sim N \left( \sum_{r|w_r = w} \mu_r L_w, \sqrt{\sum_{r|w_r = w} \sigma_r^2 L_w} \right) \quad \text{and} \quad Q_{D_w}^{-1}(y) = \min \{ S|E_{D_w}[D_w - S]^+ \leq y \}.

In every period, the retailer orders \textit{S_r} - \textit{IP_r} from the warehouse and the warehouse orders \textit{S_w} - \textit{IP_w} from the supplier where \textit{IP_r}, \textit{IP_w} are the retailer’s and warehouse’s respective inventory positions which is the sum of the on-hand and that in the pipeline vector.
Table 6: Best hyper parameters selected for each environment and method, for the benchmark RL methods. See Tables 3 and 4 for hyper parameter abbreviations.

| method setting | SAC | TD3 | PPO | A2C |
|----------------|-----|-----|-----|-----|
| 1S-3R-High     | G=0.9 LR=0.01 | G=0.9 LR=0.0003 | G=0.9 LR=0.003 VFC=1.0 | G=0.8 LR=0.003 VFC=0.5 |
| 1S-3R          | G=0.75 LR=0.003 EO=True | G=0.8 LR=0.0003 EO=False | G=0.8 LR=0.003 VFC=1.0 | G=0.8 LR=0.003 VFC=0.5 |
| 1S-10R         | G=0.8 LR=0.003 EO=True | G=0.9 LR=0.003 EO=False | G=0.8 LR=0.003 VFC=1.0 | G=0.9 LR=0.003 VFC=0.5 |
| 1S-20R         | G=0.9 LR=0.003 EO=False | G=0.75 LR=0.003 EO=False | G=0.8 LR=0.003 VFC=3.0 | G=0.9 LR=0.010 VFC=0.5 |
| 1S-2W-3R       | G=0.8 LR=0.003 EO=False | G=0.9 LR=0.003 EO=False | G=0.8 LR=0.003 VFC=1.0 | G=0.8 LR=0.003 VFC=3.0 |
| 1S-2W-3R (DS)  | G=0.9 LR=0.0003 EO=True | G=0.9 LR=0.0003 EO=True | G=0.75 LR=0.003 VFC=3.0 | G=0.8 LR=0.003 VFC=0.5 |
| 1Sinf-2W-3R    | G=0.99 LR=0.003 EO=False | G=0.99 LR=0.003 EO=False | G=0.99 LR=0.010 VFC=0.5 | G=0.99 LR=0.003 VFC=3.0 |

Parameters 1S-3R-High 1S-3R 1S-10R 1S-20R 1S-2W-3R 1S-2W-3R (DS) 1Sinf-2W-3R
Retailer demand distribution [N(2,10)] [N(2,10)] [N(2,10)] [N(2,10)] [N(2,10)] [N(2,10)] [N(2,10)]
Retailer revenue per item [50] [50] [50] [50] [50] [50] [50]
Retailer holding cost [1,2,4] [1,2,4,8] [1,2,4] [1,2,4] [1,2,4] [1,2,4,8] [1,2,4]
Supplier production per step 15 10 25 40 10 10 100
Supplier holding capacity 100 100 150 300 100 100 500
Warehouse holding cost - - - - [0.5] [0.5, 0.1] [0.5]
Warehouse holding capacity - - - - [150] [150] [150]
Spillage cost at S,W,R [10] [10] [10] [10] [10] [10] [10]
Lead time (S or W to R) [1,2,3] [1,2,3,6] [1,2,3] [1,2,3] [1,2,3,6,3,7] [1,2,3]
Lead time (S to W) - - - - - - [2]
Fixed order cost (S or W to R) [50] [50] [50] [50] [50] [50] [50]
Fixed order cost (S to W) - - - - - - [0]
Variable order cost (any link) [0] [0] [0] [0] [0] [0] [0]
Maximum order (any link) [50] [50] [50] [50] [50] [50] [50]
Initial inventory distribution (node or link) [U(0,4)] [U(0,4)] [U(0,4)] [U(0,4)] [U(0,4)] [U(0,4)] [U(0,4)]

Table 7: Environment parameters for different supply chains studied. In the 1Sinf-2W-3R setting only the non-supplier inventory pipelines are part of environment state.

| Setting       | PARL-quantile per-step reward | PARL-random per-step reward | PARL-quantile per-step train-time (s) | PARL-random per-step train-time (s) |
|---------------|------------------------------|------------------------------|--------------------------------------|-------------------------------------|
| 1S-3R-High    | 514.8 ± 5.3                  | 505.3 ± 11.0                 | 0.178 ± 0.06                         | 0.457 ± 0.26                       |
| 1S-3R         | 400.3 ± 3.3                  | 399.5 ± 2.8                  | 0.051 ± 0.01                         | 0.053 ± 0.01                       |
| 1S-10R        | 1006.3 ± 29.5                | 1005.4 ± 21.1                | 0.089 ± 0.03                         | 0.12 ± 0.07                        |
| 1S-2W-3R      | 398.3 ± 2.5                  | 395.3 ± 3.1                  | 0.051 ± 0.01                         | 0.050 ± 0.01                       |
| 1S-2W-3R (DS) | 405.4 ± 2.0                  | 398.9 ± 9.7                  | 0.044 ± 0.01                         | 0.043 ± 0.01                       |

Table 8: Comparison of quantile and random sampling in PARL.