Math Programming based Reinforcement Learning for Multi-Echelon Inventory Management

Anonymous Author(s)
Affiliation
Address
email

Abstract

Reinforcement Learning has lead to considerable break-throughs in diverse areas such as robotics, games and many others. But the application of RL to 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 51.3% and RL based methods by up to 9.58% on average across different supply chain environments.

1 Introduction

Reinforcement Learning (RL) has led to considerable breakthroughs in diverse areas such as games [1], robotics [2] 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 [3]. 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 [4, 5, 6]. 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 [7, 8], or relevant constraints are relaxed and heuristics are used to estimate feasible solutions [9, 10], or domain expertise is used to decide to approximate the state-space representation [11, 12].

The current paper takes a different approach to solving this problem. Our approach uses Neural Networks to approximate the value-to-go 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 neural network to approximate the value-to-go function along with techniques from SAA. In each iteration, the approximated 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 (9.53% on average across different settings) as well as a standard inventory management heuristic (up to 51.3% 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 approximate dynamic programming [13]. ADP methods use an approximation of the value-to-go 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-to-go 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, NN can be used to approximate the value-to-go, thereby replacing the step of feature and function selection [14, 15].

Mathematical programming based RL actor: Mathematical programming (MP) techniques have recently been used for optimizing actions in RL settings with DNN-based function approximator and large action spaces. They leverage MP to optimize a mixed-integer (linear) problem (MIP) over 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 a MP with [16, 17] also providing ideal reformulations that improve computational efficiencies with a solver. [18] propose a Q-learning framework to optimize over continuous action spaces using a combination of MP and DNN actor. [19, 14, 15] show how to use RELU-based DNN value-to-go functions to optimize combinatorial problems (e.g., vehicle routing, traveling salesman) where the immediate rewards are deterministic and the action space is vast. We extend the approaches and results to problems where immediate reward can be uncertain as in the case of inventory management problems.

RL for inventory management: Early work that show the benefits of RL for multi-echelon inventory management problems include [11, 20, 21]. There has been a recent surge in using DNN-based reinforcement learning techniques to solve supply chain problems [4, 5, 6] because the widely used base stock (s, S) threshold polices are known to be optimal only special cases (e.g., serial chain with back-ordered demands or the inability to hold demand in warehouses). See seminal works of [22, 23] and a recent review of multi-echelon inventory models studied in [24]. 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 § 4). A DNN-based actor critic method to solve the inventory management problem in [4] for the case of single node lost sales and dual sourcing settings, as well as multi-echelon settings and show improved performance in the latter setting. [5] shows how RL can be used to solve the classical beer game problem where agents in a serial supply chain compete for limited supply. More recently [6] 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 a MP-based RL actor and show the benefit over vanilla DRL approaches. These methods have the ability factor in *known* state dependent constraints as opposed to learning them.

3 PARL: Programming Actor Reinforcement Learning

We consider an infinite horizon discrete-time discounted Markov decision process (MDP) with the following representation: states $s \in \mathcal{S}$, actions $a \in \mathcal{A}(s)$, uncertain random variable $D \in \mathbb{R}^{\texttt{dim}}$ with probability distribution $P(D = d|s)$ that depends on the context state s, reward function $R(s, a, D)$, distribution over initial states β, discount factor γ 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 $\mathcal{A}(s)$ taken at state s. Then the expected return of a policy $\pi \in \Pi$ is given by $J^\pi = E_{s\sim\beta}V^\pi(s)$ where the value-to-go function is defined as $V^\pi(s) = \sum_{t=0}^{\infty} \mathbb{E}\left[\gamma^t R(s_t, a_t, D_t)|s_0 = s, \pi, P, T\right]$. The optimal policy is given by $\pi^* := \arg\max_{\pi\in\Pi} J^\pi$. The Bellman's operator $F[V](s) = \max_{a\in\mathcal{A}(s)} \mathbb{E}_{D\sim P(./s,a)}\left[R(s, a, D) + \gamma V(T(s, a, D))\right]$ over the state space is known to have a unique fixed point (i.e., to $V = FV$) at V^{π^*}. 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 $\mathcal{A}(s)$ 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 $a \in \mathcal{A}(s)$.

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: Programming Actor Reinforcement Learning (see Algorithm 1). PARL is initialized with a random policy. The initial policy is iteratively improved over epochs with a learned critic (or the value-to-go function). In epoch *j*, policy π_{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 parametrized by θ. Particularly, in every epoch, for each sample path, we also get an estimate of the cumulative reward given by $Y_n(s_0^n) = \sum_{t=1}^{T} \gamma^{t-1} R_{it}, \forall n = 1, .., N$, where s_0^n is the initial state of sample-path *n*. Note that to increase the buffer size, we also use partial sample paths. The initial states and cumulative rewards can be then passed on to a neural network which estimates the value of policy π_{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

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

Problem (1) is hard to solve because of two main reasons. First, notice that $\hat{V}^{\pi_{j-1}}$ 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 $\max_{a\in\mathcal{A}(s)} R(s, a, d) + \gamma\hat{V}_\theta^{\pi_{j-1}}(T(s, a, d))$. We describe a mathematical programming

(MP) approach to solve this problem. We begin by assuming the value-to-go V-function is a trained K-layer feed forward RELU-network with input state s satisfying the following equations:

$$z_1 = s,\ \hat{z}_k = W_{k-1} z_{k-1} + b_{k-1},\ z_k = \max\{0, \hat{z}_k\}, \forall k = 2, \cdots, K,\ \hat{V}_\theta(s) := c^T \hat{z}_K,$$

where $\theta = (c, \{(W_k, b_k)\}_{k=1}^{K-1})$ 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 $\hat{z}_k, z_k$ denotes 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 [18, 17]. 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^i, b_k^i). Assuming a bounded input $x \in [l, u]$, the output z of that neuron can be obtained with the following MP representation:

$$P(w,b,l,u) = \left\{ (x,z,y) \Big| \begin{array}{c} z \geq w^T x + b, z \geq 0, z \leq w^T x + b - M^-(1-y), z \leq M^+ y \\ x \in [l,u], y \in \{0,1\}, z \in \mathcal{R} \end{array} \right\} \quad (2)$$

where $M^+ = \max_{x \in [l,u]} w^T x + b$ and $M^- = \min_{x \in [l,u]} w^T x + b$. Let $\tilde{u}_i = u_i$ if $w_i \geq 0$ and l_i otherwise and, let $\tilde{l}_i = l_i$ if $w_i \geq 0$ and u_i otherwise. Hence $M^+ = w^T \tilde{u} + b$ and $M^- = w^T \tilde{l} + b$. Note that if $M^+ <= 0$ (or if $M^- >= 0$), 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 $\mathcal{S}$, 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 $\mathcal{A}(s)$, lend themselves in reformulating Problem (1) as a MP for any given realization of d. In Appendix 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 Sample Average Approximation (SAA) approach to solve it. Let $d_1, d_2, ..d_\eta$ denote η independent realizations of the uncertainty *D*. Then, we let

$$\hat{\pi}_j^\eta(s) = \arg \max_{a \in \mathcal{A}(s)} \frac{1}{\eta} \sum_{i=1}^{\eta} R(s, a, d_i) + \gamma \hat{V}_\theta^{\pi^\eta_{j-1}}(T(s, a, d_i)). \quad (3)$$

Problem (3) involves evaluating the objective only at sampled demand realizations. Assuming that for any η, the set of optimal actions is non empty, we show that as the number of samples, η grows, the estimated optimal action converges to the optimal action. We make this statement precise in Proposition 3.1.

Proposition 3.1 *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 π_{j-1}. Suppose $\pi_j, \hat{\pi}_j^\eta$ are the optimal policies as described in Problem (1) and its corresponding SAA approximation respectively. Then, $\forall$ s,*

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

Proposition 3.1 shows that the quality of the estimated policy improves as we increase the number of demand samples. Nevertheless, the computationally complexity of the problem also increases linearly with the number of samples: for each demand sample, we represent the DNN based value-to-go estimation using binary variables and the corresponding set of constraints.

We propose to use a weighting scheme when the uncertainty distribution $P(D = d|s)$ is known and independent across different dimensions. Let $q_1, q_2, ..q_\eta$ denote η quantiles (for example, evenly split between 0 to 1). Also let F_j & $f_j, \forall j = 1, 2.., \texttt{dim}$, denote the cumulative distribution function and the probability density function of the uncertainty D in each dimension respectively. Let $d_{ij} = F_j^{-1}(q_i)$ & $w_{ij} = f_j(q_i), \forall i = 1, 2, .., \eta, j = 1, 2.., \texttt{dim}$ denote the uncertainty samples and their corresponding probability weights. Then, a single realization of the uncertainty is a `dim` dimensional vector $d_i = [d_{i1}, .., d_{i,\texttt{dim}}]$ with associated probability weight $w_i^{pool} = w_{i1} * w_{i2} .. * w_{i,\texttt{dim}}$. With η realizations of uncertainty in each dimension, in total there are $\eta^{\texttt{dim}}$ such samples. Let $\mathcal{Q} = \{d_i, w_i^{pool}\}$ be the set of demand realizations sub sampled from this set along with the weights (based on maximum weight or other rules) such that $|\mathcal{Q}| = \eta$. Also let $w_{\mathcal{Q}} = \sum_{i \in \mathcal{Q}} w_i^{pool}$. Then Problem (3) becomes

$$\hat{\pi}_j^{\eta}(s) = \arg \max_{a \in \mathcal{A}(s)} \sum_{d_i \in \mathcal{Q}} w_i \left(R(s, a, d_i) + \gamma \hat{V}_{\theta}^{\pi_{j-1}^{\eta}}(T(s, a, d_i)) \right), \tag{4}$$

where $w_i = w_i^{pool} / w_{\mathcal{Q}}$. 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

Initialize with random actor policy π_0.
for $j \in [\mathcal{T}]$ **do**
  for (epoch) $n \in [N]$ **do**
    Play policy π_{j-1} for $T(1-\epsilon)$ and random action for ϵT steps starting with state $s_0^n \sim \beta$.
    Let $R_t^{cum,n} = \sum_{i=t}^{T} \gamma^{i-t} R_i^n$ and store tuple $\{s_t^n, R_t^{cum,n}\}$ $\forall t = 1, .., T$.
  end for
  Approximate a DNN value-to-go approximator by solving

$$\hat{V}_j = \arg \min_{\theta} \sum_{n=1}^{N} \sum_{t=1}^{T} \left(R_t^{cum,n} - f(s_t^n, \theta) \right)^2$$

  Sample η 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.
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 Λ 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^p which is either kept or distributed to other nodes. Any such distribution from node l to l' has a deterministic lead time $L_{ll'} \geq 0$ and is associated with a fixed cost $K_{ll'}$ and a variable cost $C_{ll'}$. Every node uses the inventory on-hand to fulfill local stochastic demand denoted by the r.v D_l^d 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^E. 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 $O_l \subset \Lambda \cup S^E$. 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 $U_{ll'}^L$ and $U_{ll'}^H$. There is also holding capacity at every node denoted by $\bar{U}_l$. 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:

$$V(\mathbf{I}) = \max_{x_{l'l} \in Z^+, \mathbf{U^L} \leq \mathbf{x} \leq \mathbf{U^H}} E_{\mathbf{D}} \left[R(\mathbf{I}, \mathbf{x}, \mathbf{D}) + \gamma V(\mathbf{I}') \right] \tag{5}$$

$$\text{where} \quad R(\mathbf{I}, \mathbf{x}, \mathbf{D}) = \sum_{l \in \Lambda} R_l(\mathbf{I}_l, \mathbf{x}_l, \mathbf{D}_l), \tag{6}$$

$$R_l(\mathbf{I}_l, \mathbf{x}_l, \mathbf{D}_l) = p_l \min\{D_l^d, \tilde{I}_l^0\} - \sum_{l' \in O_l} \left[K_{l'l} \mathbb{1}_{x_{l'l} > 0} + C_{l'l} x_{l'l} \right] - h_l I_l'^0, \quad \forall\, l \in \Lambda, \tag{7}$$

$$\tilde{I}_l^0 = I_l^0 + I_l^1 + D_l^p + \sum_{l' \in O_l} x_{l'l} \mathbb{1}_{L_{l'l}=0} - \sum_{\{l' \in \Lambda | l \in O_{l'}\}} x_{ll'}, \quad \forall\, l \in \Lambda, \tag{8}$$

$$I_l'^0 = \min\left\{\bar{U}_l, \left[\tilde{I}_l^0 - D_l^d\right]^+\right\}, \quad \forall\, l \in \Lambda, \tag{9}$$

$$I_l'^j = I_l^{j+1} + \sum_{l' \in O_l} x_{l'l} \mathbb{1}_{L_{l'l}=j}, \quad \forall\, 1 \le j \le \max_{l' \in O_l} L_{l'l}, l \in \Lambda. \tag{10}$$

Here $\mathbf{I}$ is the inventory pipeline vector for all nodes and the state space of the MDP, $\mathbf{x}_l$ 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), $\mathbf{I}'$ the next state defined by the transition dynamics in Eqs. (9-10) and auxiliary variables $\tilde{I}_l^0$ defined in Eq. (8). 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^0, incoming pipeline inventory I_l^1, stochastic node production D_l^p, the incoming inventory from other nodes with lead time zero and the out-going inventory from this node.

Note that the state space $\mathbf{I}$ is a collapsed state space compared to the inventory pipelines over connections between nodes as the reward $R_{tl}(.)$ 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 $C_{ll'}$ 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 Λ or be an external supplier S^E. 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.

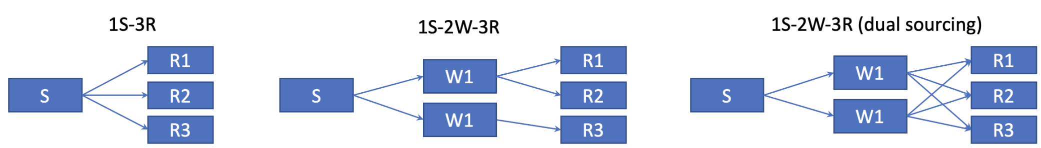


Figure 1: Example of different multi-echelon supply chain networks. In 1S-3R, a single supplier node serves a set of 3 retail nodes directly. In 1S-2W-3R, the supplier node serves the retail nodes through two warehouses. In 1S-2W-3R (dual sourcing), each retail nodes can is served by two distributors.

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.2, 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-to-go 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 [11, 4] for a hybrid model). For a single node retail node ($|\Lambda| = 1$) and one external supplier S^E, 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 [22]. 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 [25, 26] and [27, 28] prove structural results when $p/h \to \infty$. [29] 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, recently [30] propose a learning-based method to find the best base stock policy in a single node lost sales setting with regret guarantees and [31] 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 *actors* 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*, which here is the revenue from fulfillment and cost of ordering and holding, and returns the *next state* to the actor.

We consider 5 different instances of this environment for our computations based on the 2 and 3 echelon structures described in Fig. 1. We consider 3 variations of 2 echelon supplier-retailer settings: 1S-3R-High, 1S-3R, 1S-10R, where high refers to higher production capacity compared to the 1S-3R system, and 2 variations of the 3 echelon system: 1S-2W-3R and 1S-2W-3R (DS), with (DS) referring to dual sourcing. The specific details on the parameters for each of these environments are provided in Appendix A.4.

Benchmark Algorithms: We compare PARL with four state-of-the-art, widely used RL algorithms: PPO [32], TD3 [33], SAC [34], and A2C [35]; and a popularly used (s, S) base stock policy [36] for each link.

For the RL algorithms we used the tested and reliable implementations provided by Stable-Baselines3 [37], under the MIT License. We made all our environment compatible with OpenAI Gym [38] and to implement PARL we built on reference implementations of PPO provided in SpinningUp [39] (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.

In the inventory management literature[36], parametric (s, S) base stock policies are discussed for retail nodes with infinite capacity upstream supplier. In this policy, if $\mathbf{I}$ is the inventory pipeline vector for a retailer, the inventory position is defined as $IP = \sum_{i=0}^{L} I^j$, where L is the lead time from the supplier, and the order quantity is $\max\{0, S - IP\}$ as long as $IP <= s$ and 0 otherwise. 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 over order, they receive inventory proportional to the request because of the proportional fulfillment strategy. For the 3-echelon 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 leadtime 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 retail demands to find the optimal parameters for that link.

Parameter tuning and Evaluation: We perform extensive tuning of different hyper parameters (HPs) of the benchmark RL algorithms considered. We first evaluated methods over a large random grid of HP combinations to narrow down hyper parameters to a reasonable subset across our environments, in particular fixing a set of gamma values to try, fixing the representations for the observation and action spaces to continuous (interestingly discrete and multi-discrete consistently performed worse, likely because of their larger space size), using ReLU activation (consistently better or equal to tanh), fixing the network architecture to the standard used 64x64 as we did not see benefit from larger or different architectures, fixing the epoch length where applicable to 2048 steps (worked better than shorter 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 hyper-parameter combinations for each benchmark method (mainly varying gamma, learning rates, and exploration options) and ran 10 different randomly seeded modeling runs for each combination. We then computed the average accuracy per epoch (using 20 evaluation episodes) across the 10 runs for each 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 hyper parameter is the granularity of the grid search, which was set to 2 units.

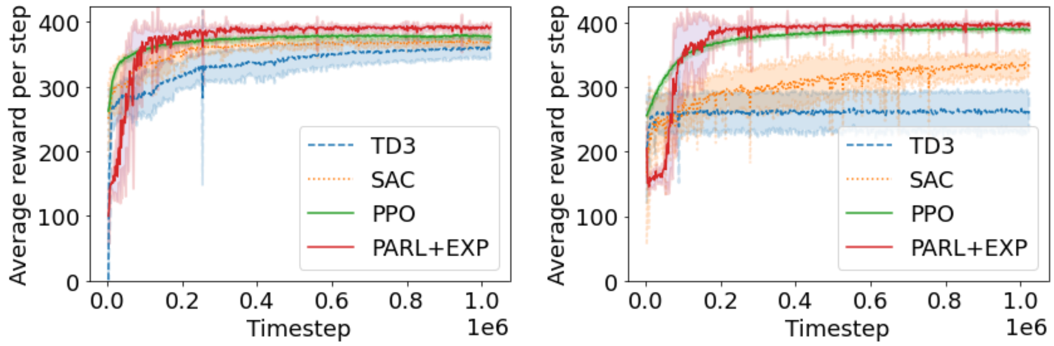
Then for all methods, given a selected best hyper-parameter combination per method and environment, to perform the evaluation we then ran 10 different training runs for each (i.e., with different random seeds). Finally we took the best epoch model according to evaluation scoring from each of those 10 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.3.

Performance: In Table 1, we present the average per step reward (over test runs) of the different algorithms and compare them to PARL. PARL outperforms all benchmark algorithms in all the five settings. Notably, the improvement is higher in supply chain settings that are more complex (1S-10R, 1S-2W-3R and 1S-2W-3R (DS)) amongst the five settings tested in the paper. While in the 10R setting, 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. In each of these settings PARL outperforms the best performing RL algorithm by 4.65% and the BS policy by 51.3% on average, across different supply chain settings.

Setting	SAC	TD3	PPO	A2C	BS	**PARL**
1S-3R-High	478.8 ± 8.5 478.3	374.7 ± 15.7 374.1	499.4 ± 5.7 500.2	490.8 ± 8.9 490.1	513.3 ± 5.9 513.0	**514.8 ± 5.3** **514.3**
1S-3R	398.0 ± 3.2 398.3	329.6 ± 45.2 311.7	397.0 ± 1.6 397.4	392.4 ± 4.4 392.87	313.7 ± 3.1 314.3	**400.3 ± 3.3** **400.8**
1S-10R	870.5 ± 68.9 905.8	744.4 ± 71.4 766.3	918.3 ± 24.7 919.2	768.1 ± 40.5 773.52	660.5 ± 2.1 659.9	**1006.3 ± 29.5** **1015.7**
1S-2W-3R	374.2 ± 3.7 375.0	361.1 ± 15.4 362.7	377.5 ± 3.7 377.5	360.2 ± 23.2 365.3	300.8 ± 5.4 302.2	**398.3 ± 2.5** **399.7**
1S-2W-3R (DS)	344.1 ± 20.6 346.1	259.3 ± 32.3 262.8	387.8 ± 5.3 388.9	327.5 ± 32.7 322.61	166.2 ± 3.8 166.4	**405.4 ± 2.0** **405.9**

Table 1: Average per-step-reward with standard deviation and (median) of different benchmark algorithms, averaged over different testing runs. PARL outperforms benchmark algorithms in all these settings.

We also analyze the rate of learning of different algorithms during training. In Figure 2, we plot the average per-step reward over training steps. On the left, we plot the outcome from the 1S-3R setting, and on the right, we plot the outcome from the 1S-2W-3R setting. We note that the average reward for the base line algorithms (TD3, SAC, PPO and A2C) is calculated without exploration while PARL's results include random exploration, with starting rate 10% and decaying every subsequent epoch (see Appendix A.3 for details). 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.



(a) 1S-2W-3R

(b) 1S-2W-3R (DS)

Figure 2: Learning curves of PARL and benchmark algorithms during training runs. PARL quickly learns a good policy and improves over benchmark algorithms.

Finally, the overall run-time of our algorithm is also of interest. In Table 2, we present the per-step run time of the algorithm in different settings during the training runs. The average per step run time is highest in the 1S-3R-High setting. This is due to the larger feasible action set in this setting. In all other settings, the run time remains below 0.10 seconds. We note that during training, we use 8 parallel environments to gather training trajectories, and use 2 CPLEX threads per environment. The run time can improve further by increasing parallelization.

Setting	Average Per-Step Run Time (PARL)	Standard Deviation
1S-3R-High	0.178	0.06
1S-3R	0.051	0.01
1S-10R	0.089	0.03
1S-2W-3R	0.051	0.01
1S-2W-3R (DS)	0.044	0.01

Table 2: Per-step run time (in seconds) of PARL over different settings.

6 Conclusions and Discussion

We consider the problem of inventory management over complex supply chain networks and develop a novel RL algorithm to solve this problem. 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. This can also be achieved by using sparse neural networks for value-to-go 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-to-go. 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

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[2] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[3] Chao Yu, Jiming Liu, and Shamim Nemati. Reinforcement learning in healthcare: A survey. *arXiv preprint arXiv:1908.08796*, 2019.

[4] Joren Gijsbrechts, Robert N Boute, Jan A Van Mieghem, and Dennis Zhang. Can deep reinforcement learning improve inventory management? performance and implementation of dual sourcing-mode problems. 2018.

[5] Afshin Oroojlooyjadid, MohammadReza 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*, 2021.

[6] Nazneen N Sultana, Hardik Meisheri, Vinita Baniwal, Somjit Nath, Balaraman Ravindran, and Harshad Khadilkar. Reinforcement learning for multi-product multi-node inventory management in supply chains. *arXiv preprint arXiv:2006.04037*, 2020.

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

[8] Linwei Xin. Understanding the performance of capped base-stock policies in lost-sales inventory models. *Operations Research*, 69(1):61–70, 2021.

[9] Sumit Kunnumkal and Huseyin Topaloglu. A duality-based relaxation and decomposition approach for inventory distribution systems. *Naval Research Logistics (NRL)*, 55(7):612–631, 2008.

[10] Awi Federgruen, C Daniel Guetta, and Garud Iyengar. Two-echelon distribution systems with random demands and storage constraints. *Naval Research Logistics (NRL)*, 65(8):594–618, 2018.

[11] Benjamin Van Roy, Dimitri P Bertsekas, Yuchun Lee, and John N Tsitsiklis. A neuro-dynamic programming approach to retailer inventory management. In *Proceedings of the 36th IEEE Conference on Decision and Control*, volume 4, pages 4052–4057. IEEE, 1997.

[12] Wenbo Chen and Huixiao Yang. A heuristic based on quadratic approximation for dual sourcing problem with general lead times and supply capacity uncertainty. *IISE Transactions*, 51(9):943–956, 2019.

[13] Warren B Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*, volume 703. John Wiley & Sons, 2007.

[14] Wouter van Heeswijk and Han La Poutré. Approximate dynamic programming with neural networks in linear discrete action spaces. *arXiv preprint arXiv:1902.09855*, 2019.

[15] Shenghe Xu, Shivendra S Panwar, Murali Kodialam, and TV Lakshman. Deep neural network approximated dynamic programming for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1684–1691, 2020.

[16] Christian Tjandraatmadja, Ross Anderson, Joey Huchette, Will Ma, Krunal Patel, and Juan Pablo Vielma. The convex relaxation barrier, revisited: Tightened single-neuron relaxations for neural network verification. *arXiv preprint arXiv:2006.14076*, 2020.

[17] Ross Anderson, Joey Huchette, Will Ma, Christian Tjandraatmadja, and Juan Pablo Vielma. Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming*, pages 1–37, 2020.

[18] Moonkyung Ryu, Yinlam Chow, Ross Anderson, Christian Tjandraatmadja, and Craig Boutilier. Caql: Continuous action q-learning. *arXiv preprint arXiv:1909.12397*, 2019.

[19] Arthur Delarue, Ross Anderson, and Christian Tjandraatmadja. Reinforcement learning with combinatorial actions: An application to vehicle routing. *Advances in Neural Information Processing Systems*, 33, 2020.

[20] Ilaria Giannoccaro and Pierpaolo Pontrandolfo. Inventory management in supply chains: a reinforcement learning approach. *International Journal of Production Economics*, 78(2):153–161, 2002.

[21] Tim Stockheim, Michael Schwind, and Wolfgang Koenig. A reinforcement learning approach for supply chain management. In *1st European Workshop on Multi-Agent Systems, Oxford, UK*, 2003.

[22] Andrew J Clark and Herbert Scarf. Optimal policies for a multi-echelon inventory problem. *Management science*, 6(4):475–490, 1960.

[23] Awi Federgruen and Paul Zipkin. Approximations of dynamic, multilocation production and inventory problems. *Management Science*, 30(1):69–84, 1984.

[24] Ton de Kok, Christopher Grob, Marco Laumanns, Stefan Minner, Jörg Rambau, and Konrad Schade. A typology and literature review on stochastic multi-echelon inventory models. *European Journal of Operational Research*, 269(3):955–983, 2018.

[25] Paul Zipkin. Old and new methods for lost-sales inventory systems. *Operations Research*, 56(5):1256–1263, 2008.

[26] Paul Zipkin. On the structure of lost-sales inventory models. *Operations research*, 56(4):937–944, 2008.

[27] Woonghee Tim Huh, Ganesh Janakiraman, John A Muckstadt, and Paat Rusmevichientong. Asymptotic optimality of order-up-to policies in lost sales inventory systems. *Management Science*, 55(3):404–420, 2009.

[28] David A Goldberg, Dmitriy A Katz-Rogozhnikov, Yingdong Lu, Mayank Sharma, and Mark S Squillante. Asymptotic optimality of constant-order policies for lost sales inventory models with large lead times. *Mathematics of Operations Research*, 41(3):898–913, 2016.

[29] Anshul Sheopuri, Ganesh Janakiraman, and Sridhar Seshadri. New policies for the stochastic inventory control problem with two supply sources. *Operations research*, 58(3):734–745, 2010.

[30] Shipra Agrawal and Randy Jia. 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*, pages 743–744, 2019.

[31] Mohammad Pirhooshyaran and Lawrence V Snyder. Simultaneous decision making for stochastic multi-echelon inventory optimization with deep neural networks as decision makers. *arXiv preprint arXiv:2006.05608*, 2020.

[32] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[33] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1582–1591, 2018.

[34] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1861–1870. PMLR, 10–15 Jul 2018.

[35] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[36] Lawrence V Snyder and Zuo-Jun Max Shen. *Fundamentals of supply chain theory*. John Wiley & Sons, 2019.

[37] Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable Baselines3. `https://github.com/DLR-RM/stable-baselines3`, 2019.

[38] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016.

[39] Joshua Achiam. Spinning Up in Deep Reinforcement Learning. `https://github.com/openai/spinningup`, 2018.

[40] Alexander Shapiro. Monte carlo sampling methods. *Handbooks in operations research and management science*, 10:353–425, 2003.

[41] Alexander Shapiro, Darinka Dentcheva, and Andrzej Ruszczyński. *Lectures on stochastic programming: modeling and theory*. SIAM, 2014.

[42] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[43] Kazuyuki Hara, Daisuke Saito, and Hayaru Shouno. Analysis of function of rectified linear unit used in deep learning. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2015.