Online Appendix
Deep Policy Iteration with Integer Programming for Inventory Management

Appendix A: Table of notation and block diagram for PARL

Symbol	Explanation
s	State in the state space $\mathcal{S}$.
a	Action in the action space $\mathcal{A}(s)$, associated with state s.
D	Uncertain random variable representing demand, taking values in $\mathbb{R}^{\texttt{dim}}$.
$P(D=d \mid s)$	Probability distribution of D given the context state s.
$R(s,a,D)$	Reward function, dependent on the state s, action a, and demand D.
β	Distribution over initial states.
γ	Discount factor for future rewards.
s'	Next state, resulting from the transition dynamics $\mathcal{T}(s,a,d)$.
π	A stationary policy, a distribution over actions given state s.
J^π	Expected return of a policy π, based on initial state distribution β and value function $V^\pi(s)$.
$V^\pi(s)$	Value function under policy π, sum of discounted rewards starting from state s.
π^*	Optimal policy that maximizes the long-term expected discounted reward.
$\mathbb{E}_D$	Expectation taken over the distribution of demand D.
θ	Parameters of the neural network representing the value-to-go function.
$R_t^{cum,n}$	Cumulative discounted reward from time t in sample path n, under policy π_{j-1}.
η	Number of independent realizations of the uncertainty D used in the SAA approach.
$\hat{V}_\theta^\pi(s)$	Approximation of the value function at state s under policy π, parameterized by θ.
$\hat{\pi}_j^\eta(s)$	Policy obtained by solving the SAA approximation problem at iteration j for state s.
Λ	Set of nodes in the supply chain network.
l	Index for nodes in the supply chain network (warehouses, distribution centers, retail stores).
D_l^p	Random variable denoting inventory produced at node l.
D_l^d	Random variable denoting demand generated at node l.
O_l	Set of upstream nodes that can ship to node l.
$L_{ll'}$	Deterministic lead time for trans-shipment from node l to node l'.
$K_{ll'}$	Fixed cost associated with trans-shipment from node l to node l'.
$C_{ll'}$	Variable cost associated with trans-shipment from node l to node l'.
h_l	Holding cost per inventory unit at node l.
p_l	Profit (or revenue) per inventory unit sold at node l.
I_l^0	On-hand inventory at node l.
$\texttt{tsc}_l$	Trans-shipment cost for node l.
$\tilde{I}_l^0$	Intermediate on-hand inventory at node l.
$\texttt{rs}_l$	Revenue from sales for node l.
$\texttt{hsc}_l$	Holding-and-salvage costs for node l.
U_l	Storage capacity of node l.
$x_{l'l}$	Trans-shipment decision, inventory shipped from node l' to l.

Table EC.1 Mathematical Notations and Their Explanations

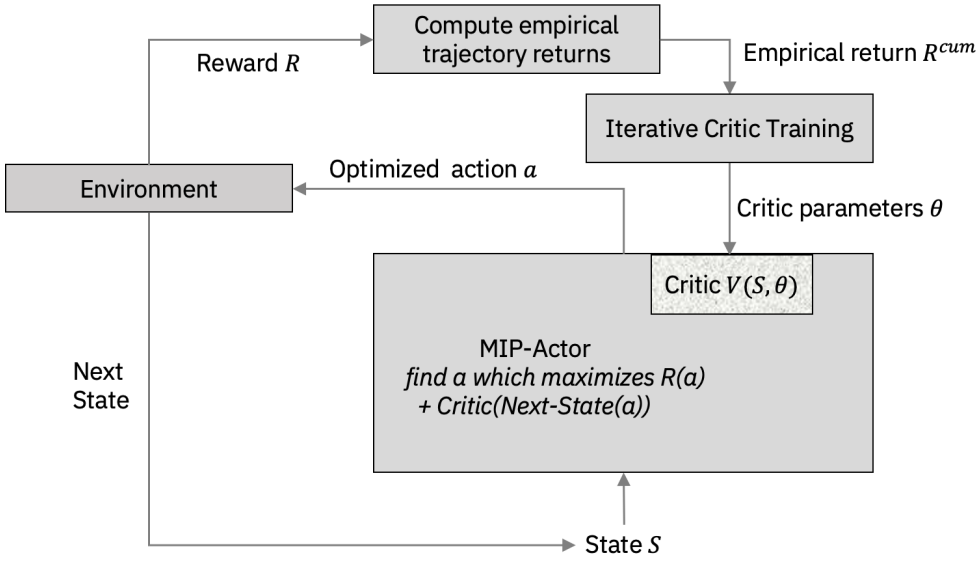


Figure EC.1 Block diagram illustration of the PARL algorithm

Appendix B: Proof of Proposition 1

Recall that our objective is to prove the following result:

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

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

To prove this result, we introduce some more notation. Consider the following problem:

$$\min_{x\in\mathcal{X}} \Big\{ f(x) := \mathbb{E}[F(x,\varepsilon)] \Big\},$$

where $\mathcal{X}$ is non-empty closed subset of R^d, ε is a random variable with finite-support. Also let $\varepsilon_1, .., \varepsilon_N$ denote N independent realizations of the random variable ε and define the Sample Average Approximation of the true problem as

$$\min_{x\in\mathcal{X}} \Big\{ \hat{f}_N(x) := \frac{1}{N}\sum_{i=1}^{N} F(x,\varepsilon_i) \Big\}.$$

Also let v^* and S^* to be the optimal value and optimal solution of the true problem and $\hat{V}_N$ and $\hat{S}_N$ to be the optimal solution for the SAA problem. Finally, let $\mathcal{D}(S_1, S_2)$ denote the discrepancy between two sets, S_1 and S_2.

Our main result uses the following two key results from the SAA literature:

THEOREM EC.1 **(Theorem 5.3 of Shapiro et al. (2014))**. *Suppose that there exists a compact set $\mathcal{C} \subset \mathbb{R}^n$ such that (i) the set of optimal solutions of the true problem is non-empty and is contained in $\mathcal{C}$, (ii) the function $f(x)$ is finite valued and continuous on $\mathcal{C}$, (iii) $\hat{f}_N(x)$ converges to $f(x)$ w.p. 1 as $N\to\infty$, uniformly in $x\in\mathcal{C}$, (iv) for N large enough the set $\hat{S}_N$ is non-empty and $\hat{S}_N \subset \mathcal{C}$. Then, $\hat{v}_N \to v^*$ and $\mathcal{D}(\hat{S}_N, S^*) \to 0$ w.p. 1 as $N\to\infty$.*

PROPOSITION EC.1 **(Proposition 7 of Shapiro (2003))**. *Let $\mathcal{C}$ be a nonempty compact subset of $\mathbb{R}^n$ and suppose that: (i) for almost every ε, $F(x,\varepsilon)$ is continuous on $\mathcal{C}$, (ii) $F(x,\varepsilon)$, $x \in \mathcal{C}$, is dominated by an integrable function, (iii) the sample is iid. Then the expected value function f(x) is finite valued and continuous on $\mathcal{C}$, and $\hat{f}_N(x)$ converges to $f(x)$ w.p. 1 as $N \to \infty$, uniformly in $x \in \mathcal{C}$.*

Proof of Proposition 1: Recall that

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

and its corresponding SAA version is given by

$$\hat{\pi}_j^\eta(s) = \arg\max_{a \in \mathcal{A}(s)} \frac{1}{\eta} \sum_{i=1}^{\eta} R(s,a,d_i) + \gamma \hat{V}_\theta^{\pi_{j-1}^\eta}(\mathcal{T}(s,a,d_i)). \tag{EC.2}$$

Let $\hat{f}_\eta^s(a) := \frac{1}{\eta}\sum_{i=1}^{\eta} g(s,a,d_i),\ \forall s$, where $g(s,a,d) = R(s,a,d) + \gamma \hat{V}_\theta^{\pi_{j-1}}(\mathcal{T}(s,a,d))$. we can then leverage results of Theorem EC.1 as long as $g(s,a,d)$ satisfies the conditions stated in Theorem EC.1.

Hence, we need to prove (i) the set of optimal solutions of (EC.1) is non-empty and is contained in $\mathcal{C}$, (ii) the function $\mathbb{E}[g(s,a,D)]$ is finite valued and continuous on $\mathcal{C}$, (iii) $\hat{f}_\eta^s(a)$ converges to $\mathbb{E}[g(s,a,D)]$ w.p. 1 as $\eta \to \infty$, uniformly for $a \in \mathcal{C}$, (iv) for η large enough, the set $\hat{S}_\eta$ is non-empty and $\hat{S}_\eta \subset \mathcal{C}$.

We start by focusing on property (iii) and show that $\hat{f}_\eta^s(a)$ uniformly converges to $\mathbb{E}[g(s,a,D)]$ with probability 1. Consequently, to prove this result, we prove two main properties of $g(s,a,d)$: (p-i) $g(s,a,d)$ is *continuous* in a for almost every $d \in D$, and (p-ii) $g(s,a,d)$ is dominated by an *integrable function*. To prove (p-ii), we show that $g(s,a,d) \le C < \infty$ w.p. 1 $\forall a \in \mathcal{A}(s)$. 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 properties (p-i) and (p-ii). 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 $\hat{V}_\theta(\mathcal{T}(s,a,d))$ with $\theta = (c, \{(W_k, b_k)\}_{k=1}^{K-1})$ denoting the parameters of the K-layer ReLU-network. As $\mathcal{T}(s,a,d)$ is continuous and $\hat{V}_\theta(s)$ is continuous, $\hat{V}_\theta(\mathcal{T}(s,a,d))$ is continuous which proves (p-i). Next, note that $\mathcal{T}(s,a,d)$ lies in a bounded space for any realization of the uncertainty d. Furthermore, since the parameters of the NN (θ) 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 which proves (p-ii). Then, following Proposition EC.1, we have uniform convergence of $g^\eta(s,a,d)$ to $\mathbb{E}[g(s,a,D)]$ w.p. 1. This proves property (iii). Properties (i), (ii) and (iv) are a direct consequence of the assumptions we make on the solution of Problem (EC.2). In particular, for all s the set of feasible actions is a bounded polyhedron $\mathcal{A}(s)$ and that for any η, 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 et al. (2015) for a discussion on the techniques used for extending the analysis to this setting.

Appendix C: Simplified example to demonstrate PARL's application in the inventory replenishment context

In this section, we discuss a simplified supply chain example to demonstrate how the PARL framework is applied to solve the inventory replenishment problem.

Supply chain: We consider a single-echelon supply chain with lost-sales where a single producer serves two retailers (see Figure EC.2a). The two retailers have identical demand but are heterogeneous with respect to the lead time as well as the holding cost. While the first retailer has a lead time of 1 and holding cost of \$1/unit/time, the second retailer has a lead time of 2 and holding cost of \$2/unit/time. Both retailers have a holding capacity of 50 units, a fixed order cost of \$50/order and finally earn a revenue of \$50/unit. Finally, the producer has no holding cost, produces 70 units/time and has a holding capacity of 70 units.

The state-space of this supply chain is the on-hand and pipeline inventory of each retailer and the supplier. It is a six-dimensional vector $\mathbf{I} = [s_1, s_2, s_3, s_4, s_5, s_6]$ where s_1 represents the on-hand inventory of the produces, the next two positions contain the on-hand and pipeline inventory of retailer one and the last three positions contain the on-hand and pipeline inventory of retailer two. Now consider any time t such that the initial state is s, the order decision is (x_1, x_2) and the demand realization is (d_1, d_2). Our approach considers splitting the total reward into two parts: immediate reward (from this time step) and the value-to-go of future rewards (generated from state transition due to current action and the subsequent actions) which is estimated using a neural network. Consider a simplified setting where the future reward is estimated using a neural network with an input layer, a single hidden layer with two neurons each, and the output layer (See Figure EC.2b). The current state $\mathbf{I}$, action a and demand realization d leads to state transition from $\mathbf{I}$ to $\mathbf{I}'$ whose value can be estimated using the NN. In particular, $\mathbf{I}'$ is a six-dimensional input vector that is passed first through the neurons of the input layer, then to the intermediate layer, and finally transformed to the output which is the estimated value-to-go. To describe the corresponding IP formulation, consider the first neuron of the input layer. The NN is pre-trained. Hence each neuron of the input layer has corresponding weights (6 dimensional weights w) and bias (scalar b) and uses the ReLU activation. We denote by (w_{11}, b_{11}) and (w_{12}, b_{12}), the weight and bias parameters of these neurons in the input layer. Then, passing the input $\mathbf{I}' \in [l, u]$ to these neurons, the output from these neurons is given by z^*_{11} and z^*_{12} where

$$
\begin{aligned}
&(z^*_{11}, y^*_{11}) = \arg\max_{z,y} \;\; 0 \\
&z \geq w_{11}^T \mathbf{I}' + b_{11},\; z \geq 0,\; z \leq w_{11}^T \mathbf{I}' + b_{11} - M^-(1-y),\; z \leq M^+ y,\; y \in \{0,1\}\,.
\end{aligned}
$$

Here, $[l, u]$ are upper and lower bounds on the state and M^+ and M^- are big Ms that can be pre-calculated (see §2.2 of the manuscript). To see the intuition about this formulation, note that for any scalar input $u \in \mathbb{R}$, the ReLU activation function is given by $g(u) := \max\{0, u\}$. Hence, if $w_{11}^T \mathbf{I}' + b_{11}$ is positive, we want z^*_{11} to equal $w_{11}^T \mathbf{I}' + b_{11}$ and 0 otherwise. In fact, the output of the system of equations is always a singleton. It is easy to check that the output of the IP above ensures these outputs. Similarly, for the second neuron in the input layer, we have that

$$
\begin{aligned}
&(z^*_{12}, y^*_{12}) = \arg\max_{z,y} \;\; 0 \\
&z \geq w_{12}^T \mathbf{I}' + b_{12},\; z \geq 0,\; z \leq w_{12}^T \mathbf{I}' + b_{12} - M^-(1-y),\; z \leq M^+ y,\; y \quad \in \{0,1\}\,.
\end{aligned}
$$

The bounded output from the input layer (z_{11}^*, z_{12}^*) becomes an input to the neurons of the second layer (denoted by NN_{21} and NN_{22} in the figure). We can use similar IP formulation to pass the output from the previous layer (input layer in this case) to this layer. For example, for NN_{21}, we have that

$$(z_{21}^*, y_{21}^*) = \arg\max_{z,y} \ \ 0$$
$$z \geq w_{21}^T[z_{11}^*, z_{12}^*] + b_{21},\ z \geq 0,\ z \leq w_{21}^T[z_{11}^*, z_{12}^*] + b_{21} - M^-(1-y),\ z \leq M^+ y,\ y \in \{0,1\}\,.$$

Similarly, for NN_{22}, we have that

$$(z_{22}^*, y_{22}^*) = \arg\max_{z,y} \ \ 0$$
$$z \geq w_{22}^T[z_{11}^*, z_{12}^*] + b_{22},\ z \geq 0,\ z \leq w_{22}^T[z_{11}^*, z_{12}^*] + b_{22} - M^-(1-y),\ z \leq M^+ y,\ y \in \{0,1\}\,.$$

We let $\mathtt{P}(w,b,x)$ denote the constraints of the IP for a neuron with weights w, bias b and input x to concisely represent the constraint sets in each of these problems. Finally, the output from the hidden layer is passed on to the output layer for final value-to-go estimation of the input state s':

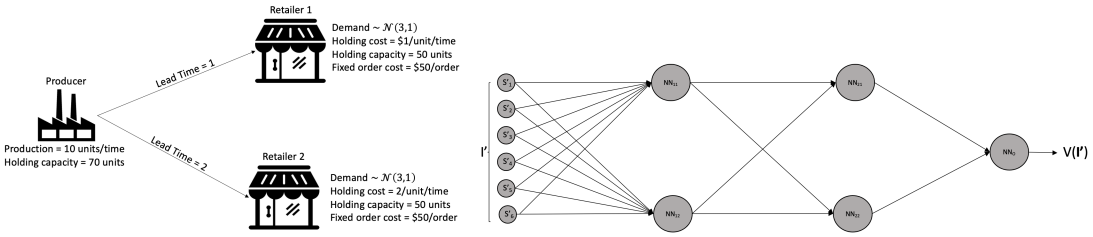
$$\begin{aligned} V(\mathbf{I}') &= c^T[z_{21}^*, z_{22}^*] + b_o \\ s.t.\ (z_{1j}^*, y_{1j}^*) &\in \mathtt{P}(w_{1j}, b_{1j}, \mathbf{I}')\ \forall j \in [1,2] \\ (z_{2j}^*, y_{2j}^*) &\in \mathtt{P}(w_{2j}, b_{2j}, [z_{11}^*, z_{12}^*])\ \forall j \in [1,2] \end{aligned} \tag{EC.3}$$

where $c \in \mathbb{R}^2$ and $b_o \in \mathbb{R}$ are pre-trained parameters corresponding to the output layer.

Note that the input $\mathbf{I}'$ is directly a function of the current action (along with the current state s and the demand realization d). Let $\mathbf{I}' := \mathcal{T}(\mathbf{I}, \mathbf{x}, \mathbf{d})$ denote the next state given the current state $\mathbf{I}$, the action $\mathbf{x}$ and the demand realization $\mathbf{d}$. Then, the optimal action maximizes both the immediate reward as well as the future cumulative discounted rewards from this action (estimated using the NN described above). In particular, given the current state $\mathbf{I}$, we solve the following optimization problem per time step

$$\mathbf{x}^* = \max_{\mathbf{x} \in \mathcal{L}(\mathbf{I})} \frac{1}{2} \sum_{i=1}^{2} \left[\overset{*}{R}(\mathbf{I}, \mathbf{x}, \mathbf{d}_i) + \gamma V(\mathbf{I}') \right], \tag{EC.4}$$

where we consider two demand realizations d_1 and d_2 to approximate the total reward for different actions, $\overset{*}{R}(\mathbf{I}, \mathbf{x}, \mathbf{d}_i)$ denotes the immediate reward (cost) based on inventory decision $\mathbf{x}$ (for example fixed order cost, holding cost, revenue from sales etc.); $V(\mathcal{T}(\mathbf{I}, \mathbf{x}, \mathbf{d}_i))$ is the value to go estimate (from Eq. EC.3); $\mathcal{L}(\mathbf{I})$ denotes the set of feasible inventory actions given the current state $\mathbf{I}$, and γ is the pre-selected discounting parameter. Both $\mathcal{L}(\mathbf{I})$ and $\mathbf{I}'$ can be represented using math programming constraints. Furthermore, the set of constraints discussed above can ensure that the value-to-go in the objective function is evaluated correctly. Finally, the optimal action can be computed by solving this IP.



(a) 1S-2R (b) Simple value-to-go neural network

Figure EC.2 **Simplified supply chain network. On the left, we consider a single echelon setting with a single producer and two retailers with lost sales. On the right, a simplified neural network with an input layer (2 neurons), a hidden layer (2 neurons) and an output layer.**

Appendix D: More details on benchmark algorithms and parameter settings

D.1. Supply chain parameters in different settings

Parameters	1S-3R-High	1S-3R	1S-10R	1S-20R	1S-2W-3R	1S-2W-3R (DS)	$1S^{inf}$-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]	[1,2,4,8]	[1,2,4,8]	[1,2,4]	[1,2,4]	[1,2,4]
Retailer holding capacity	[50]	[50]	[50]	[50]	[50]	[50]	[50]
Supplier production qty 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]	[1,2,3]	[1,2,3]	[1,2,3]	[(1,5),(2,6),(3,7)]	[1,2,3]
Lead time (S to W)	-	-	-	-	[2]	[2]	[2]
Fixed order cost (S or W to R)	[50]	[50]	[50]	[50]	[50]	[50]	[50]
Fixed order cost (S to W)	-	-	-	-	[0]	[0]	[0]
Variable order cost (any link)	[0]	[0]	[0]	[0]	[0]	[0]	[20] for S-W
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 EC.2 Environment parameters for different supply chains studied. In all the settings, we assume deterministic, constant per-period production and that the variability is only in the demand. The parameters are provided by 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 implies that 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 implies that 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 μ and standard deviation σ 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 parts 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. Next, we discuss different benchmark algorithms that we tested in this paper.

D.2. Base Stock heuristic policy

The seminal work of Scarf (1960) shows the optimality of the parametric (s, S) base stock policies for retail nodes with infinite capacity upstream supplier in backordered demand settings. This optimality holds even in the case with fixed costs and constant lead times. In this policy, if $\mathbf{I}$ is the inventory pipeline vector for a firm, 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.

Due to the popularity of these policies, we implement a heuristic base stock policy for the multi-echelon networks we study as follows. At the high level, we construct a base stock policy for every link assuming

the upstream entity's supply is unconstrained. For the 2 echelon $1\texttt{S}-n\texttt{R}$ environments, we identify the best base stock policy via grid search for each link using a $1\texttt{S}-1\texttt{R}$ environment with unconstrained supply. The reason we do a grid search is because we are in the lost sales setting. Note that we observe that the constrained supply setting also yields very similar results and hence restrict ourselves to the unconstrained supply setting. For the 3-echelon environments, we use the same strategy for the $\texttt{W}-\texttt{R}$ 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 $\texttt{S}-\texttt{W}$ links we use environments that treat the warehouse as a retailer with demand equal to the sum of the downstream (lead time) retail demands to find the optimal parameters for that link.

D.3. Decomposition-Aggregation (DA) heuristic (Rong et al. 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 $S_r = F_{D_r}^{-1}[\frac{b_r}{b_r+h_r}]$ where $F_{D_r}^{-1}$ is the inverse cumulative demand distribution (cdf) of the random variable $D_r \sim (\mu_r(L_r+1), \sigma_r\sqrt{L_r+1})$. 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 inventory and that in the pipeline vector.

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

For each such sample path, we compute $S_r = F_{D_r}^{-1}[\frac{b_r+h_{w_r}}{b_r+h_r}]$ where $F_{D_r}^{-1}$ is the inverse cdf of the random variable $D_r \sim N(\mu_r(L_r+1), \sigma_r\sqrt{L_r+1})$. 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 $q_{w_r} = F\left[0.5F^{-1}\left[\frac{b_r}{b_r+h_r}\right] + 0.5F^{-1}\left[\frac{b_r}{b_r+h_{w_r}}\right]\right]$ where F and F^{-1} 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 $S_{w_r} = F_{D_{w_r}}^{-1}[q_{w_r}]$ where D_{w_r} is distributed as $\text{N}(\mu_r(L_r+L_w+1), \sigma_r\sqrt{L_r+L_w+1})$. An expected shortfall is computed which is $Q_{D_{w_r r}}(s_{w_r}) = E_{D_{w_r r}}[D_{w_r r} - s_{w_r}]^+$ where $s_{w_r} = S_{w_r} - S_r$.

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

In every period, the retailer orders $S_r - IP_r$ from the warehouse and the warehouse orders $S_w - IP_w$ from the supplier where IP_r, IP_w are the retailer's and warehouse's respective inventory positions which is the sum of the on-hand inventory and that in the pipeline vector.

D.4. Parameters and policy for the $1\texttt{S}^{\text{inf}}$-1R environment

The $1\texttt{S}^{\text{inf}}$-1R environment is a setting with back ordered demand and no-fixed costs. The lead time (L) is 4, the holding cost (h) is 1.8 and the back order penalty (b) is 7. The demand (D) is assumed to be normal $N[\mu, \sigma]$ where $\mu = 5$ and $\sigma = 0.8$. In this setting the optimal policy is an order-up to policy with closed form

solution $F_D^{-1}\left(\frac{b}{b+h}\right)$ which is 26.48. As the environment only allows ordering integer quantities, we tested both 26 and 27 order up-to levels and 27 outperformed on average across 10 (identical) model runs each with 20 test runs with trajectory length of 10K steps.

D.5. Benchmarking PARL and PPO on serial acyclic chains

In this section, we discuss results from additional numerical experiments to test the performance of the proposed PARL algorithm on serial and general-acyclic-non-tree networks. We implement the optimal Clark and Scarf solution[8] for these settings and compare PARL and PPO's performance against the optimal solution. We test the $1\mathtt{S}^{\inf}-1\mathtt{W}-1\mathtt{W}-1\mathtt{R}$, and $1\mathtt{S}^{\inf}-1\mathtt{W}-1\mathtt{W}-1\mathtt{W}-1\mathtt{R}$ settings with a lead time of 1 between all nodes. The backorder cost is set to 20. The holding costs at the warehouses and retailers are set to 1, 2, and 4, and 1, 2, 3, and 5 respectively. Finally, following standard practice, we also apply holding costs for the in-transit inventory. In Figure EC.3, we present results from these experiments. We find similar insights as before: both PARL and PPO's performance is comparable to the optimal policy, with an optimality gap of 0-5% in these settings.

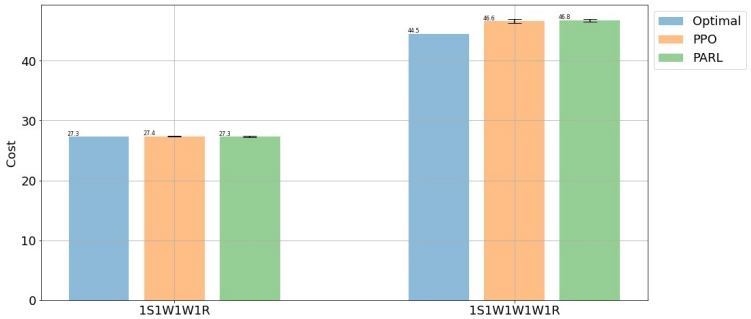
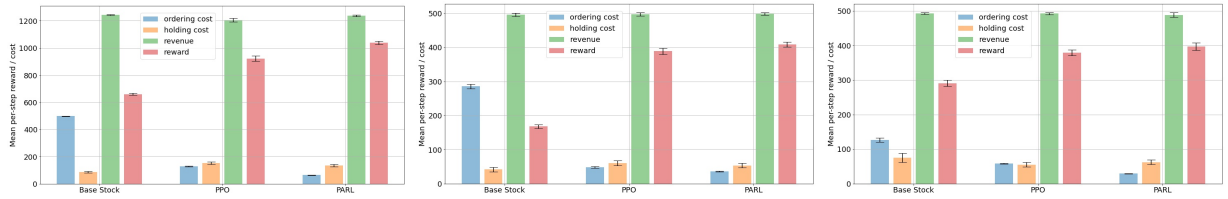


Figure EC.3 **Benchmarking against Clark-and-Scarf policies for serial acyclic supply chains. We consider two different acyclic supply chains ($1\mathtt{S}^{\inf}-1\mathtt{W}-1\mathtt{R}$, and $1\mathtt{S}^{\inf}-1\mathtt{W}-1\mathtt{W}-1\mathtt{R}$) and benchmark PARL and PPO against the Clark-and-Scarf based optimal policy. Both PARL and PPO's performance is comparable to the optimal policy's performance in this setting.**

[8] Using exhaustive search over all possible order-upto integral solutions

D.6. Cost and reward comparison for other settings



(a) 1S-10R (b) 1S-2W-3R (DS) (c) 1S-2W-3R

Figure EC.4 **Breakdown of rewards in test across BS, PPO and PARL algorithms for settings not discussed in the main text. Note that ordering costs include fixed and variable costs of ordering, revenue refers to the revenue earned from sales and reward refers to the revenue net costs incurred (see §3 and Table EC.2 for more details).**

D.7. Comparison of quantile and random sampling in PARL

In many settings, the decision maker might have extra information on the underlying uncertainty D. In these cases, one can use specialized weighting schemes to estimate the expected value to go for different actions, given a state. For example, consider the case when the uncertainty distribution $P(D=d)$ 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 $d_{ij} = F_j^{-1}(q_i)$ & $\texttt{w}_{ij} = f_j(q_i), \forall i = 1, 2, .., \eta, j = 1, 2.., \texttt{dim}$ denote the uncertainty samples and their corresponding probability weights. Then, a single realization of the uncertainty is a $\texttt{dim}$ dimensional vector $d_i = [d_{i1}, .., d_{i,\texttt{dim}}]$ with associated probability weight $\texttt{w}_i^{pool} = \texttt{w}_{i1} * \texttt{w}_{i2} .. * \texttt{w}_{i,\texttt{dim}}$. With η realizations of uncertainty in each dimension, in total there are $\eta^{\texttt{dim}}$ such samples. Let $\mathcal{Q} = \{d_i, \texttt{w}_i^{pool}\}$ 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 $\texttt{w}_\mathcal{Q} = \sum_{i \in \mathcal{Q}} \texttt{w}_i^{pool}$. Then Problem (4) becomes

$$\hat{\pi}_j^\eta(s) = \arg\max_{a \in \mathcal{A}(s)} \sum_{d_i \in \mathcal{Q}} \texttt{w}_i \left(R(s,a,d_i) + \gamma \hat{V}_\theta^{\pi_{j-1}^\eta}(\mathcal{T}(s,a,d_i)) \right), \tag{EC.5}$$

where $\texttt{w}_i = w_i^{pool} / \texttt{w}_\mathcal{Q}$. 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. Here we compare the use of quantile sampling and random sampling to generate realizations of the uncertainty in (4). 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 EC.3, 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.

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 ± 5.3 514.3	505.3 ± 11.0 505.1	0.178 ± 0.06	0.457 ± 0.26
1S-3R	400.3 ± 3.3 400.8	399.5 ± 2.8 400.8	0.051 ± 0.01	0.053 ± 0.01
1S-10R	1006.3 ± 29.5 1015.7	1005.4 ± 21.1 1007.3	0.089 ± 0.03	0.12 ± 0.07
1S-2W-3R	398.3 ± 2.5 399.7	395.3 ± 3.1 395.9	0.051 ± 0.01	0.050 ± 0.01
1S-2W-3R (DS)	405.4 ± 2.0 405.9	398.9 ± 9.7 402.0	0.044 ± 0.01	0.043 ± 0.01

Table EC.3 Comparison of quantile and random sampling in PARL.

Appendix E: Parameter Tuning Details

Table EC.4 Best hyper parameters selected for each environment and method, for the benchmark RL methods. See Tables EC.5 and EC.6 for hyper parameter abbreviations.

method setting	SAC	TD3	PPO	A2C
1S-3R-High	G=0.9 LR=0.01 EO=True	G=0.9 LR=0.0003 EO=False	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=0.0003 EO=True	G=0.8 LR=0.003 VFC=1.0	G=0.9 LR=0.003 VFC=1.0
1S-20R	G=0.9 LR=0.003 EO=False	G=0.75 LR=0.0003 EO=False	G=0.9 LR=0.010 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=0.0003 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
$1S^{inf}$-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

Table EC.5 Tuning hyper parameters and additional fixed hyper parameters for PPO and A2C - we vary γ, learning rate, and value function coefficient - resulting in 36 hyper parameter combinations

Hyper Parameters for PPO and A2C	**Value(s)**
Discount Factor - γ *(G)*	0.99, 0.9, 0.80, 0.75
Learning rate *(LR)*	0.01, 0.003, 0.0003
Value function coefficient (in loss) *(VFC)*	0.5, 1.0, 3.0
Number of steps to run per update (epoch length)	2048
Max gradient norm (for clipping)	0.5
GAE lambda (trade-off bias vs. variance for Generalized Advantage Estimator)	0.95 (PPO) and 1.0 (A2C) (defaults)
Number of epochs to optimize surrogate loss (internal train iterations per update - PPO only)	20
KL divergence threshold for policy update early stopping per epoch (PPO only)	0.15 ("target_kl"=0.1)
Clip range (PPO only)	0.2
RMSprop epsilon (A2C only)	1e-05

Table EC.6 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 - γ *(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 EC.7 Fixed set of hyper-parameters used for all methods

Batch size	Net arch. (hidden layers per net)	Activation	State representation	Action representation	Epoch length
64	64x64	ReLU	continuous (normalized)	continuous (normalized)	2048

Appendix F: Additional experiments to test the scalability of the proposed method

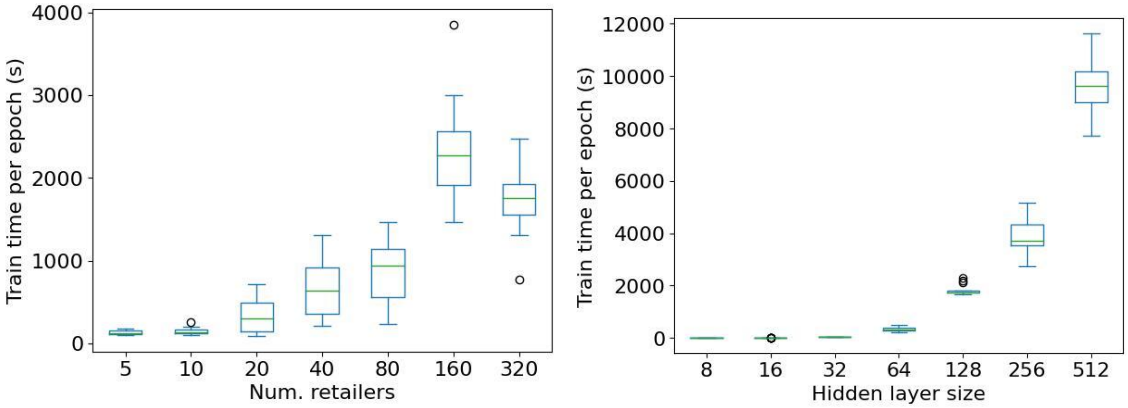
In this section, we further expand on the computational experiments to test the scalability of the proposed framework. We consider two specific dimensions of the problem: (i) number of retailers in the supply chain network and (ii) number of hidden layers in the value approximating NN. In Figure EC.5 we plot the results from these experiments[9]. As expected, as the size of the supply chain or the NN grows, so does the runtime of the algorithm. In Figure EC.5a, we scale the number of retailers from 5 to up to 320 retailers in the supply chain[10] and analyze the per epoch runtime for different supply chain network sizes. We find that the runtime scales nearly linearly from 4 seconds per epoch (for the smallest supply chain) to up to 2000 seconds for the largest network. The effect of scaling NN on the runtime is more pronounced. In Figure EC.5b we scale the number of neuron in the hidden layer from 8 to 512 [11]. The runtime increases nearly exponentially as the number of neurons in the hidden layer increases. This is as expected since scaling the number of neurons, scales the number of connections in the dense neural network quadratically which translates to bigger IP formulations. Finally, given these insights, we note that moderate to large sized supply chain networks (300

[9] We ran these timing calculations on a few different but same-configured linux VMs with Ubuntu 22.04 OS. Each VM has Intel(R) Xeon(R) Gold 6140 CPUs @ 2.30GHz (i.e., Intel processors) and has 42 cpus and 188GB. When we ran the experiments, for each run we used 8 processes to parallelize running different trajectories (episodes) and we used 2 threads for solving each MIP. For each timing experiment, for each value on the x-axis (size) we ran the environment and training for 10 epochs (with exploration amount set at the average across a normal 500 epoch run), and averaged the total time over the 10 epochs to get the average time per epoch. We repeated this 15 times for each setting as well - to account for differences in randomization and timing for running on the virtual machines (for which timing can change due to network load and traffic). We report the statistics of the 15 runs for each setting.

[10] Note that the supply chain network is restricted to the 1S-MR setting. Furthermore, the Neural Network configuration is the same as the rest of the paper: it has two fully connected hidden layers of 64 neurons each.

[11] In this case, we fix the supply chain network to be the 1S-3R and scale the number of neurons in each hidden layer of the neural network

retailers) with moderately sized NN approximator (64 neurons per hidden layer, 2 fully connected hidden layers) can be trained in approximately 10 days in our computational setup. Since all this training can happen offline and the learned model can be directly used for inventory replenishment decisions in real-time. To decrease the compute time further, we note two possible directions for further improving the computations speed: (i) Any episode / trajectory / hyper parameter computation can be done completely in parallel, given enough compute resources - as a training batch is made up of some number of episodes, this parallelization avoids the cost of additional episodes in a batch. (ii) Solving the IP could also be sped up by using more CPUs/parallel processes or threads as well. We used 2 threads in our experiments, and more threads can speed up the process further. Nevertheless, both these directions require additional computational resources that might limit wider adaptability.



(a) 1S-mR (b) 1S-3R

Figure EC.5 Run-time analysis as we scale the supply chain network and the neural network size

We also run additional experiments to test how the proposed algorithm scales when using open-source integer programming solvers instead on CPLEX. In particular, we replicate the IP formulation for neural network value evaluation using both SCIP (Bolusani et al. 2024) and Pulp-CBC (Lougee-Heimer 2003). Note that the run-time of the single-step action problem in PARL is driven by the run-time of this evaluation step. Hence, isolating this run-time across different open-source solvers can give us insights on how the per-step action run-time could change if we use open source solvers in our problem. In Figure EC.6, we compare the solver run-time of both SCIP as well as PuLP-CBC as we increase the size of the neurons in the fully connected hidden layers of the neural network from 8 to 128. As expected, CPLEX performs better than both the open-source solvers. Furthermore, as the complexity of the NN increases, so does the gap between the run-time from CPLEX and the open source solvers. Nevertheless, the difference in runtime between CPLEX and SCIP for moderately sized NNs (up to 64 neurons per layer) is marginal.

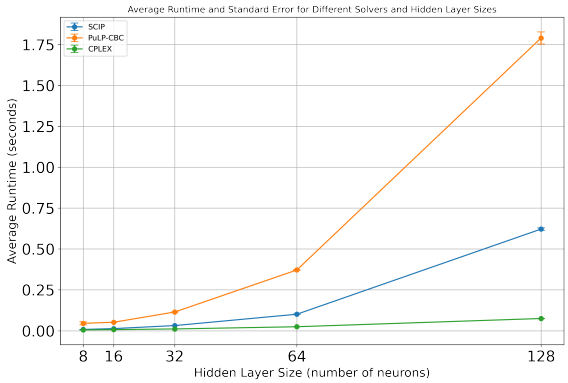


Figure EC.6 Run-time analysis as we change the underlying IP solver.

Appendix G: Open Source Package Usage

```
[conf_type]
conf_type = graph # Specify network via graph or list

[env_params]
env_type = pdr # pdr (define producers, distributors, and retailers) or 1sMr
state_rep = N # Normalized continuous state representation
action_rep = MD # Multi-discrete action representation
quant = 1 # Order action quantization amount
reset_max_entity_inv = 4 # Max initial inventory randomly generated on reset
reset_max_connection_inv = 4
back_order = False #Set to True for back order at retailers setting instead of lost sales


[supply_chain_general_params]
max_order_action = 50 # Maximum order amount

# Next, for each of producers, distributors and retailers, define list of IDs and
    associated lists of settings for each
# Note: default distribution for all entities is (truncated, rounded) stationary Gaussian
     - other distributions can be easily specified and plugged in (Poisson, non-
    stationary versions, and inventory-dependent are already available options)

[supply_chain_producer_params]
id_list = P1
prod_daily_prod_avg_list = 10 # Mean production per producer
prod_daily_prod_std_list = 0. # Std. dev. of production per producer
holding_cost_list = 0
holding_capacity_list = 100
overorder_penalty_list = 0
max_start_inv = -1 # if below 0, set equal to prod_holding_cap

[supply_chain_distributor_params]
id_list = D1, D2
holding_cost_list = 0.5, 0.1
holding_capacity_list = 150, 150
```

```
overorder_penalty_list = 10, 10
max_start_inv = 60, 60

[supply_chain_retailer_params]
id_list = R1, R2, R3
demand_avg_list = 2, 2, 2 # Mean demand per retailer
demand_std_list = 10, 10, 10 # Std. dev. of demand per retailer
revenue_list = 50, 50, 50
holding_cost_list = 1, 2, 4
overorder_penalty_list = 10, 10, 10
holding_capacity_list = 50, 50, 50
max_start_inv = 12, 12, 12
backorder_penalty_list = 0, 0, 0

[supply_chain_connection_params]
# Define connections and their parameters (costs and lead times) between defined entities
upstream_id_list = P1, P1, D1, D1, D1, D2, D2, D2
downstream_id_list = D1, D2, R1, R2, R3, R1, R2, R3
L_list = 2, 2, 1, 2, 3, 5, 6, 7 # Specify lead times for each connection
order_cost_per_item_list = 0, 0, 0, 0, 0, 0, 0, 0
order_cost_fixed_list = 0, 0, 50, 50, 50, 50, 50, 50
max_start_inv = 6, 6, 6, 6, 6, 6, 6, 6
```

Listing 1: 1S-2W-3R with dual sourcing environment configuration example

```
from supply_chain_rl.run.env_setup_from_cfg import get_env_fun
from stable_baselines3 import PPO
from supply_chain_rl.model.parl import PARL

#Instantiate the environment with normalized continuous action representations
env_function = get_env_fun("env.cfg", action_rep="n")
env = env_function()

#Instantiate a baseline DRL model or PARL model - randomly initialized
# model = PPO(policy="MlpPolicy", env=env)
model = PARL(env=env)

#Fit the model to the env
model.learn(total_timesteps=1e6)

#Evaluate model for some number of steps and log costs and rewards
#Reset environment, send action from model to env to get reward and next state
state = env.reset()
env.logging_enable()

for step in range(256):
    action, _ = model.predict(state)
    state, reward, done, info = env.step(action)
    print(step, reward)

#Export log of evaluation to a file:
env.logging_write_json("trajectory_log.json")
```

Listing 2: Example Python code using the library to load a supply chain environment and train and evaluate PARL or a baseline RL algorithm on the environment.