ORIE 6590 Project Report: Inventory Control with Lead Times

Wenchang Zhu
wz368@cornell.edu

Zhi Liu
zl724@cornell.edu

1 Introduction and Model Basics

In this project, we study an inventory control problem with lost sales and long lead times. Such problems are traditionally considered intractable due to the curse of dimensionality. Recent papers such as (Xin and Goldberg, 2016) showed that under the infinite horizon setting, the performance of the constant order policy converges exponentially fast when lead time tends to infinity. For the scope of this project, we aim at using reinforcement learning to generate satisfactory policies, and make comparisons with existing ones, especially in the cases where constant-order policies fail to get good performance.

The code used in this project can be found in the GitHub repository here.

1.1 Model Setup

In this section we formally define the lost-sales inventory optimization problem. To make comparisons easier, we consider the same model introduced in (Xin and Goldberg, 2016), where we have i.i.d non-negative demands following exponential distribution and non-negative orders at every time period.

Let $\{D_t, t \geq 1\}$ be a sequence of independent and identically distributed (i.i.d.) demand realizations, distributed as the non-negative random variable (r.v.) D with distribution $\mathcal{D}$, which we assume to have finite mean, and strictly positive variance. Let $L \geq 1$ be the deterministic lead time, and $h, p(> 0)$ be the unit holding cost and lost-sales penalty, respectively.

In addition, let I_t denote the on-hand inventory, and $\mathbf{x}_t = (x_{1,t}, \ldots, x_{L,t})$ denote the pipeline vector of orders placed but not yet delivered, at the beginning of time period t, where $x_{i,t}$ is the order to be received in period $i + t - 1$. Our state space consists of all the possible values of (I_t, x_t). The ordered sequence of events in period t is then as follows.

- A new amount of inventory $x_{1,t}$ is delivered and added to the on-hand inventory.
- A new order is placed.
- The demand D_t is realized.
- Costs for period t are incurred, and the on-hand inventory and pipeline vector are updated.

Note that the on-hand inventory is updated according to $I_{t+1} = \max(0, I_t + x_{1,t} - D_t)$, and the pipeline vector is updated such that (s.t.) $x_{1,t}$ is removed, $x_{i,t+1}$ is set equal to $x_{i+1,t}$ for $i \in [1, L-1]$, and $x_{L,t+1}$ is set equal to the new order placed.

A policy π consists of a sequence of measurable maps $\{f_t^\pi, t \geq 1\}$, where each f_t^π is a function with domain $\mathbb{N}^{L+1}$ and range $\mathbb{N}$. In that case, for a given policy π, the order placed in period t equals $f_t^\pi(\mathbf{x}_t, I_t)$. Note that the constraint on the domain and range of the policy function is introduced for tractability. Normally, they should be $\mathbb{R}^{+,L+1}$ and $\mathbb{R}^+$, as we will discuss in the next subsection.

We let $C_t \triangleq h(I_t + x_{1,t} - D_t)^+ + p(I_t + x_{1,t} - D_t)^-$ denote the sum of the holding cost and lost-sales penalty in time period t, where $x^+ \triangleq \max(x, 0), x^- \triangleq \max(-x, 0)$.

As we will be interested in minimizing long-run average costs, and for simplicity, we suppose that the initial conditions are such that the initial inventory is 0 , and the initial pipeline vector is empty, i.e. $I_1 = 0, \mathbf{x}_1 = \mathbf{0}$.

For a given policy $\pi \in \Pi$, we let C_t^π denote the corresponding cost incurred in period t, and

$$C(\pi) \triangleq \lim_{T\to\infty} \frac{1}{T}\mathbb{E}\left[\sum_{t=1}^{T} C_t^\pi\right]$$

denote the long-run average cost incurred by π. Then the objective in this inventory control problem is simply

$$\min_{\pi\in\Pi} C(\pi)$$

where Π denote the family of all admissible policies.

1.2 MDP parameters

In order to solve our problems using RL algorithms, we formulate the problem as a MDP and introduce our parameters as follows:

- State space: $\mathcal{S} = \mathbb{R}_+^{L+1}$.
- Action space: $\mathcal{A} = \mathbb{R}_+$
- Reward function: $r(I_t, x_t, a) = -h(I_t + x_{1,t} - D_t)^+ - p(I_t + x_{1,t} - D_t)^-$.
- Transitions: From time t to $t+1$, $I_{t+1} = \max(0, I_t + x_{1,t} - D_t)$, $x_{i,t+1} = x_{i+1,t}$, for $i = 1, \ldots, L-1$. $x_{L,t+1}$ is set to be the new action taken at this step.

2 Model Implementation, Verification and Validation

In this section, we introduce the details of our implementation of the model, and we further verify and validate that the model is consistent with theoretical results.

2.1 Implementation

We implemented our model using the `gym` package. Noting that the state space and action space are both continuous, we implemented them as `box` objects, where the state space is a $L+1$ dimensional `box` with lower bound 0 and upper bound infinity, and the action space is a 1 dimensional `box` with

lower bound 0 and upper bound yet to specify. Upper bound for the action space is important to the performance of RL algorithms, and we will discuss choosing an appropriate upper bound later.

The transition of this MDP mainly depends on generating a demand at each period. We implemented the demand at each period with an exponential random variable generator with $\lambda = 1$. This choice of parameter is justified since (Xin and Goldberg, 2016) noted that due to the scaling properties of exponential distribution, the choice of λ does not affect the ratio of performance between two policies; on the other hand, a moderately sized parameter, such as $\lambda = 1$, is easier to work with.

2.2 Verification and Validation

The key ingredients of our model are the state space, the action space and the transition. By using the `box` object to model the state and action space the key properties of them are verified, and the transition is verified with the above description about realizing the demand.

To validate our model, we make use of some important results from (Xin and Goldberg, 2016). The first result is on the optimal performance of the constant order policy. (Xin and Goldberg, 2016) showed that, when the demand is exponential with rate λ, the optimal constant order policy has average cost

$$C\left(\pi_{r_\infty}\right) = \lambda^{-1}(\sqrt{h(2p+h)} - h)$$

where π_{r_∞} denotes the policy that makes a constant order of r_∞, the optimal amount, during each period, and the value of r_∞ is calculated as

$$r_\infty = \lambda^{-1}\left(1 - \sqrt{\frac{h}{2p+h}}\right)$$

To make use of this result, we implemented constant order policies of the above amount, and evaluated their performances. Then we compared them with the theoretical performance. The results are gathered in the table below.

We can observe that, our implemented model matches the theoretical performance with high precision, showing that it is consistent with the theoretical model in this regard. We also note that, when p is large, the optimal constant order approaches λ^{-1}, which the authors showed will be unstable in the sense that the average cost diverges. This explains why the performance of the optimal constant order policy for large p tend to be less stable than that for smaller p.

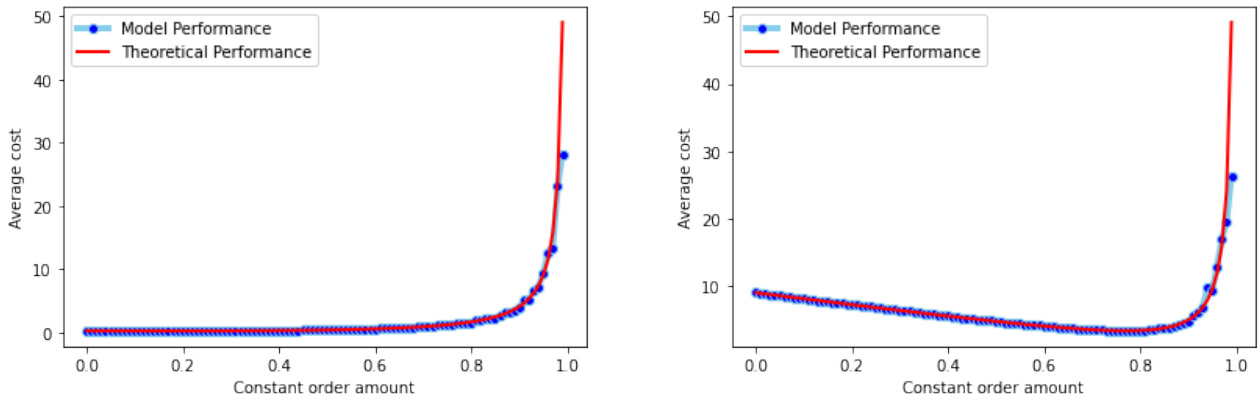
Table 1: Performance of optimal constant-order policy with our model implementation

	L=1	L=4	L=10	L=20	Theoretical Perf.
p=0.25	0.225±0.002	0.225±0.002	0.222±0.002	0.225±0.003	0.2247
p=1	0.726±0.008	0.727±0.013	0.726±0.01	0.734±0.012	0.7321
p=4	1.966±0.049	1.992±0.024	2.034±0.057	1.996±0.079	2.0000
p=9	3.345±0.132	3.351±0.145	3.344±0.096	3.365±0.127	3.3589
p=39	7.539±0.481	7.891±0.582	7.833±0.53	7.955±0.489	7.8882
p=99	13.006±1.964	13.72±1.66	12.982±1.233	14.163±0.846	13.1067
	L=30	L=50	L=70	L=100	Theoretical Perf.
p=0.25	0.223±0.003	0.225±0.003	0.224±0.003	0.225±0.003	0.2247
p=1	0.732±0.01	0.732±0.013	0.735±0.012	0.733±0.009	0.7321
p=4	1.982±0.044	1.998±0.048	1.986±0.061	2.03±0.043	2.0000
p=9	3.366±0.154	3.398±0.065	3.347±0.121	3.338±0.194	3.3589
p=39	8.111±0.408	7.837±0.428	7.745±0.391	7.862±0.631	7.8882
p=99	13.655±0.905	13.464±1.93	12.919±1.08	13.742±3.661	13.1067

To further validate our model, we use another result on the theoretical performance of constant order policy, that is its average cost, as a function of the constant order amount r, is found to be

$$C(\pi_r) = h\frac{r^2\lambda}{2(1-r\lambda)} + p\lambda^{-1} - pr$$

Below we show two plots of the performance generated from our model compared with the theoretical performance, under different values of p and L.



(a) $p = 0.25, L = 30$ (b) $p = 9, L = 10$

Figure 1: Performance of constant-order policies on our model compared with theoretical results

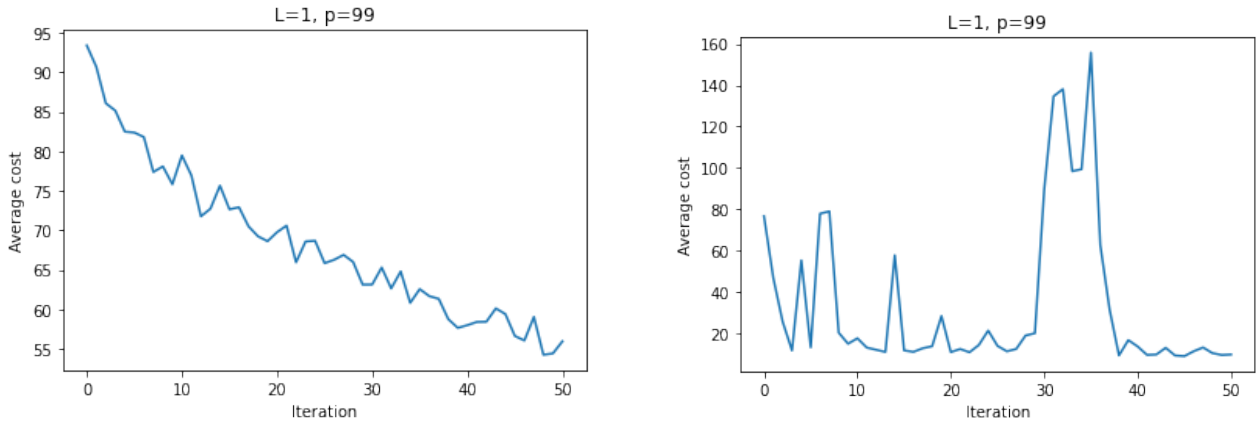
The figures show that our implementation aligns with the theoretical results well for the range of constant-order policies.

The above two results validate that our implementation is consistent with the theoretical model, and provides a good foundation for us to move forward.

3 Reinforcement Learning Algorithm and Parameters

3.1 Choice of Algorithm

We used the Proximal Policy Optimization(PPO) algorithm from Stable Baselines3 (Hill et al., 2018) package. The reason for this choise is that our model consists of both continuous action space and continuous state space, which limits our choice of algorithms; in addition, after initial performance comparison of PPO with Advantage Actor Critic (A2C), both with default parameters, we find that PPO with default outperforms A2C both in terms of training time and the resulting policy. The following is an example with model parameters $L = 1, p = 99$, note the difference in scale of the Y-axis.



(a) A2C with default parameters

(b) PPO with default parameters

Figure 2: Performance of A2C and PPO with default parameters

In light of this, we decided to fine-tune the PPO algorithm to try and improve its performance. The next subsection goes into details about the parameters we choose and the reason behind them.

3.2 PPO

We used the variant PPO-clip which updates policies via

$$\theta_{k+1} = \arg\max_{\theta} \mathop{\mathrm{E}}_{s,a\sim\pi_{\theta_k}} \left[c\left(s, a, \theta_k, \theta\right) \right],$$

and c is given by

$$c\left(s, a, \theta_k, \theta\right) = \min\left(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_k}(a \mid s)} A^{\pi_{\theta_k}}(s, a), \quad \text{clip}\left(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_k}(a \mid s)}, 1-\epsilon, 1+\epsilon \right) A^{\pi_{\theta_k}}(s, a) \right).$$

Some parameters of this algorithm is specially chosen, which include:

- Policy: we use the `MlpPolicy` model which implements Actor-Critic, using a multilayer perceptron for both the policy (Actor) network and the value (Critic) network. We will introduce our policy initialization in detail in section 3.3.1.

- Max action allowed: We noticed that, if we set $\mathcal{A} = \mathbb{R}^+$, it may happen that PPO algorithm diverges as iterations increase; if the set of action is too small, for example $\mathcal{A} = [0.0, 5.0]$, then performance is compromised as we may want to order more than this amount in some situations. After some trials we restricted it to a moderately sized space ($\mathcal{A} = [0, 20]$) for actions to make learning easier and to ensure good performance.

- `n_envs`: We used `make_vec_env` and generated 4 environments for muliti-processing.

- `learning_rate`: The optimizer's learning rate is set to be 0.0003.

- `n_steps`: We set the number of steps to run for each environment per update to be 2048. We tried to relate this with L and did some experiments to decide the optimal number. We found that this default choice balances run-time with stability of the sample path best.

- `batch_size`: The minibatch size is set to be 64.

- `n_epochs`: Number of epoch when optimizing the surrogate loss in each iteration. We set it to be 10.

- `clip_range`: The clipping parameter ϵ is set to be 0.2.

- `gamma`: sine we are in an average-cost setting, we set $\gamma = 1$.

- `gae_lambda`: Factor for trade-off of bias vs variance for Generalized Advantage Estimator. We set it to be 0.95 when $L \in \{1, 4, 10, 20\}$, and set it to be 0.99 when $L \in \{30, 50, 70, 100\}$. A more detailed discussion is given in section 3.3.2.

- `vf_coef`: Value function coefficient for the loss calculation is set to be 0.5.

- `max_grad_norm`: The maximum value for the gradient clipping is set to be 0.5.

Under each parameter pair L and p, we run the PPO algorithm until generated `iteration` exceeds 50. Note that at each iteration we run `n_envs` $\times$ `n_steps` $= 8192$ steps, so the PPO algorithm stops after `total_timesteps`$= 409600$. We noticed that is enough for PPO algorithm to get satisfactory results for our inventory problem and our algorithm usually converge within 20 iterations.

To evaluate our policy, we generate several long enough episodes $\tau_n = (s_0^n, a_0^n, s_1^n, a_1^n, \ldots, s_T^n, a_T^n)$ and estimate the cost by $\frac{1}{N(T+1)} \sum_{n=1}^{N} \sum_{t=0}^{T} r(s_t^n)$. By strong law of large number, this finite sum converge to the long-time average cost almost surely when T tends to infinity. This justifies our evaluation. The standard error is given by the different episodes.

3.3 Insights for Learning

3.3.1 Policy Initialization

For the policy and value network, we did not use any feature extractor layer before them, as we believe that the state of the model itself is well-structured with low dimension (maximum 100-dimensional) and contains valuable information. The way we initialize the policy is by setting the weights of action net to be 0 and setting the bias of action net to be a constant so that we get a state-independent initial policy which is near 'optimal constant' policy(see table 2). This will ensure that the initial policy is a slightly randomized policy that mimics an optimal constant-order policy

and most importantly, is stable. As for the value network, the default random parameterization is used.

We did experiments using both default initialization and near constant-order initialization and found that although default initialization managed to start PPO from a stable policy, applying near constant-order initialization allows us to converge to an ideal cost in fewer iterations(see Figure 3). We see that with custom initialization we managed to get good policies within only 3 iterations. This reduces a lot of computational cost.

Table 2: network initialization parameters

p	0.25	1	4	9	39	99
`action_net.bias`	0.1	0.2	0.3	0.4	0.5	0.6

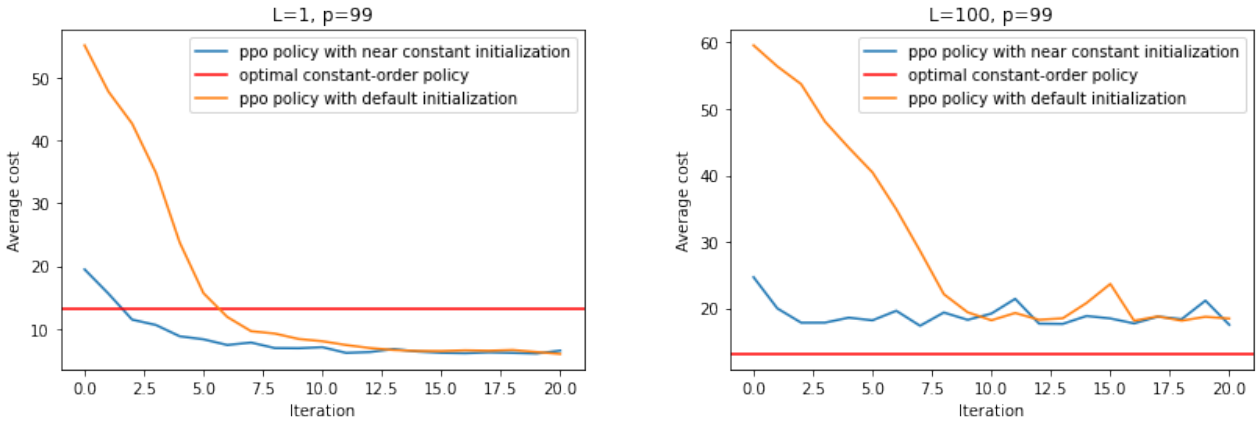


Figure 3: Performance of PPO with and without custom networks

3.3.2 Balance of Variance and Bias

While estimating the advantage function, we used the Generalized Advantage Estimator (GAE). Instead of using the standard value estimator $\hat{h}^{(s^*)}(s) = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{\sigma^n(s^*)-1}\left(r\left(s_t^n\right)-\eta\right)$, we add a discount factor `gae_lambda` to reduce the variance. The estimator then becomes $\hat{h}^{(s^*)}(s) = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{\sigma^n(s^*)-1}\texttt{gae\_lambda}^t\left(r\left(s_t^n\right)-\eta\right)$. When `gae_lambda` tends to 0, we have no variance but large bias. When `gae_lambda` tends to 1, we have no bias but large variance. We observe that when L is small, the default value `gae_lambda`$= 0.95$ gives good performance, so we stick to `gae_lambda`$=$ 0.95 when $L \in \{1, 4, 10, 20\}$. However, when state space becomes larger, `gae_lambda`$= 0.95$ is not enough for good performance, which might because of large state space needs the value estimation to have smaller bias. Based on this observations, we set `gae_lambda`$= 0.99$ when $L \in \{30, 50, 70, 100\}$.

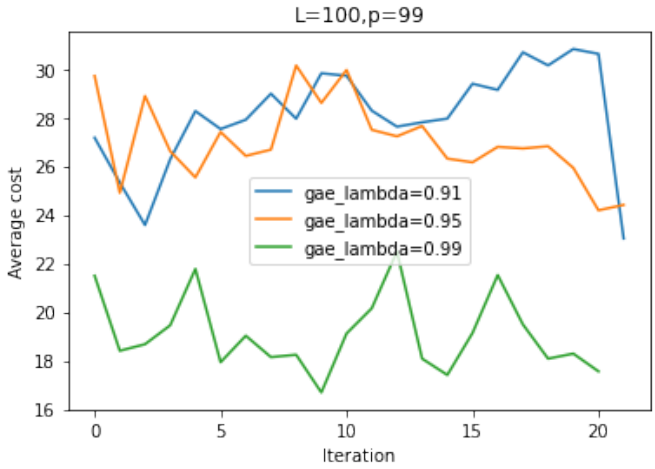


Figure 4: PPO performance with different `gae_lambda`

4 Results and Discussion

4.1 Theoretical Results for Constant-order Policy

In this section, we recall the performance guarantee for the optimal constant-order policy introduced in (Xin and Goldberg, 2016). As mentioned in Section 2.2, when the demand is exponential with rate λ, the optimal constant order policy has average cost $C(\pi_{r_\infty}) = \lambda^{-1}(\sqrt{h(2p+h)}-h)$. Furthermore, for all $L \geq 1$, they proved the following bound:

$$\frac{C(\pi_{r_\infty})}{OPT(L)} \leq 1 + \left(\tau_{p,h} + \left(\tau_{p,h}^{-1} - 1\right)(e(L+1))^{-1}\right)\left((1-\gamma_{p,h})\log\left(1+ph^{-1}\right)\right)^{-1}\gamma_{p,h}^{L+1}, \qquad (1)$$

here $\tau_{p,h} \triangleq \sqrt{\frac{h}{2p+h}}$, and $\gamma_{p,h} \triangleq (1-\tau_{p,h})\exp(\tau_{p,h})$. Evaluating (1) they got the following table:

Table 3: When $h = 1$, values of (1) under different p and L.

Evaluations of (1)	L=1	L=4	L=10	L=20	L=30	L=50	L=70	L=100
p=0.25	2.13	1.08	1.00	1.00	1.00	1.00	1.00	1.00
p=1	3.36	1.89	1.15	1.01	1.00	1.00	1.00	1.00
p=4	6.42	3.99	2.62	1.72	1.34	1.08	1.02	1.00
p=9	12.26	6.77	4.43	3.12	2.45	1.73	1.38	1.15
p=39	62.26	27.60	14.86	9.62	7.62	5.75	4.75	3.81
p=99	204.5	85.21	41.77	24.43	18.20	12.92	10.49	8.49

We will compare our results obtained by RL algorithms with the above guarantees in the following sections.

4.2 Results for PPO

In this section, we demonstrate our results of PPO algorithm with both default parameters and our tuned parameters. We observe that our results outperform the optimal constant-order policies under a number of parameters, especially in the setting where L is small but p is large.

Table 4: When h = 1, PPO values under different p and L with default parameters.

PPO values	L=1	L=4	L=10	L=20	L=30	L=50	L=70	L=100	$C(\pi_{r_\infty})$
p=0.25	0.243	0.242	0.243	0.240	0.240	0.247	0.247	0.245	0.225
p=1	**0.725**	0.750	0.765	0.818	0.884	0.890	0.913	0.928	0.732
p=4	**1.751**	**1.930**	**1.985**	2.136	2.510	2.679	2.795	2.852	2.000
p=9	**2.646**	**2.961**	**3.292**	3.547	4.060	4.580	4.682	5.123	3.359
p=39	**4.306**	**5.387**	**6.458**	7.707	9.828	11.207	13.839	22.616	7.888
p=99	**5.263**	**6.706**	**9.151**	**11.894**	16.496	19.592	32.900	53.861	13.107

Table 5: When h = 1, PPO values under different p and L with tuned parameters.

PPO values	L=1	L=4	L=10	L=20	L=30	L=50	L=70	L=100	$C(\pi_{r_\infty})$
p=0.25	**0.224**	**0.224**	0.225	0.227	0.249	0.254	0.269	0.263	0.225
p=1	**0.725**	**0.731**	**0.732**	0.736	0.885	0.880	0.924	0.942	0.732
p=4	**1.834**	**1.924**	**1.996**	2.058	2.452	2.451	2.577	2.520	2.000
p=9	**2.710**	**2.968**	**3.145**	**3.340**	4.084	4.210	4.342	4.307	3.359
p=39	**4.287**	**5.366**	**6.443**	**6.708**	9.295	10.028	10.183	10.360	7.888
p=99	**5.275**	**5.676**	**8.987**	**11.587**	14.396	16.010	17.261	16.661	13.107

Table 6: When $h = 1$, the ratios of average cost of best constant order policy and PPO policy.

$C(\pi_{r_\infty})/C_{\text{PPO}}$	L=1	L=4	L=10	L=20
p=0.25	1.003679295	1.001242377	0.9992347017	0.9882196751
p=1	1.009702969	1.00111972	1.000498585	0.9950083025
p=4	1.090476645	1.041003027	1.001750057	0.9714338818
p=9	1.239246016	1.131701445	1.067925794	1.005473829
p=39	1.839808933	1.469787083	1.224236178	1.175890839
p=99	2.484440606	2.309203652	1.459110191	1.131174623
	L=30	L=50	L=70	L=100
p=0.25	0.9019487007	0.8839522965	0.834186548	0.8543548244
p=1	0.8276651711	0.8263235866	0.7919293799	0.7765815466
p=4	0.8154597115	0.8159881453	0.7762356993	0.793447394
p=9	0.8223879509	0.7977734771	0.7736202402	0.779887117
p=39	0.8486359505	0.7866035222	0.774617104	0.761343877
p=99	0.9103926294	0.8186213222	0.7592882639	0.7866477175

Table 5 shows the mean values of the average cost under different p and L obtained by PPO with default settings and the average cost under optimal constant-order policies.

Based on the insights and careful tuning of PPO parameters described in section 3.2, another set of performance is obtained in table 5. In the above tables, we highlight the cases where our policy outperforms the best constant order policy in bold characters. Additionally, we compare our policy to the best constant order policy by calculating the ratio of the average cost, which is shown as in table 6. Note that if this ratio is greater than 1 it means our policy is better than the best constant order policy.

To further demonstrate our policy, we plotted the actions taken with $L = 1$ and various p in figure

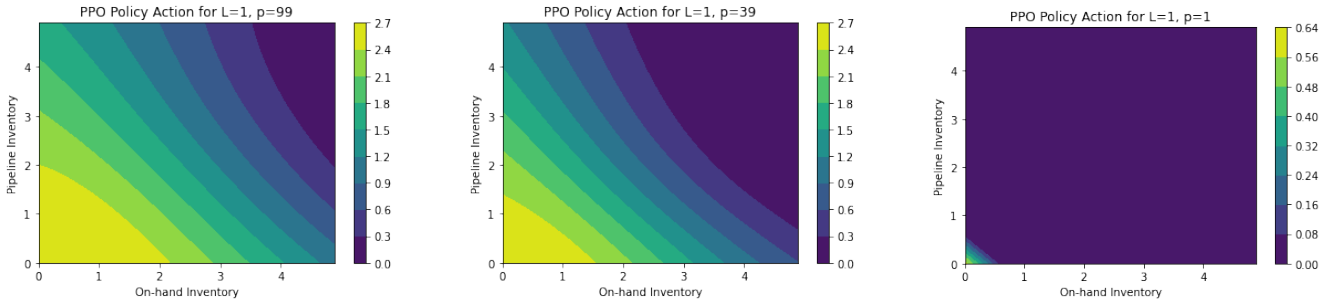


Figure 5: The actions taken by PPO policies for $L = 1$ and different p values.

It seems very natural: for large p, the policy will order more than the average demand (1 unit) even when the inventory on hand and in the pipeline are in large quantities, just to avoid this large penalty; for small p, however, the policy dictates that only a small amount of order will be placed, after all, the holding cost is the same as the penalty, so there is no point in hoarding inventory as to avoid being penalized.

4.3 Discussions

Based on the results, we observe a strong characteristics of our method, which is that it performs relatively well on cases with smaller lead times, but when lead times get larger, this cleverer policy obtained through PPO generally cannot compare with the simpler, yet very stable constant order policy. This aligns with the intuition: when the lead time is small, a more clever policy may use the current state to improve the order amount, for example, ordering less if there is more order in the pipeline, thus countering the randomness of the demand; when the lead time is large, however, the amount that we order is not going to arrive soon, and it is harder to predict what randomness we may encounter during this period. It is often believed that when prediction is hard, a simple policy that guarantees an adequate worst-case performance will beat a more clever policy that tries to make prediction.

We can thus give a guide for solving lost-sales inventory problems with lead time. When lead time is not large, we can use PPO to decide a policy. However, when lead time becomes larger, it might be better to apply constant-order policies, which require less computation yet give lower costs than PPO.

References

Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. 2018. Stable Baselines. `https://github.com/hill-a/stable-baselines`.

Linwei Xin and David A. Goldberg. 2016. Optimality Gap of Constant-Order Policies Decays Exponentially in the Lead Time for Lost Sales Models. *Operations Research* 64, 6 (2016), 1556–1565.