Reinforcement Learning for Supply Chain Optimization in Beer Production

Ziheng Wang [1]

Abstract

This report explores reinforcement learning solutions for the Supply Chain Optimization in Beer Production. Starting from the given DQN baseline, we first optimize key training parameters to reach a an average episode reward of 453.83. To explore alternatives, we develop a separate value iteration–based method, which significantly outperform the DQN baseline, achieving a reward of **464.10**. Furthermore, we modify the environment to reflect realistic upstream-downstream dependencies. Under this setting, providing the full environment state to the agent leads to performance improvement.

1. Baseline DQN and Parameter Optimization

We build upon the Deep Q-Network (DQN) algorithm (Mnih et al., 2015) as our baseline, using the implementation[1] as the starting point for development. On top of this framework, we introduce the following modifications to improve stability and performance:

- **Expanded evaluation episodes:** We increase the number of evaluation episodes for the test agent from the default setting to 10,000. This significantly reduces variance in the estimated average episode reward and provides a more robust measure of the policy's true performance.
- **Extended training with early stopping:** The number of training episodes is extended to allow for better convergence. To avoid overfitting and excessive computation, we implement an early stopping mechanism: the model is evaluated every 1,000 steps, and training is terminated if no improvement is observed across three consecutive evaluations. The best-performing policy is saved.
- **Exploration rate analysis:** We investigate the effect of the exploration rate ε in the ε-greedy policy. A small ε may limit exploration, causing the agent to converge prematurely to suboptimal policies, while a large ε can result in excessive random actions and unstable learning. We test several values of ε and report three metrics: average episode reward, service rate (i.e., the ratio of demand fulfilled to total demand), and the iteration at which the best test result occurs.

[1] https://github.com/paperusername/coursebeergame

Table 1. Effect of exploration rate ε

ε	Reward	Service (%)	Best Iter
0.01	442.7	90.1	4000
0.08	446.6	91.6	7000
0.10	**453.8**	90.3	6000
0.12	438.2	90.9	4000
0.15	432.3	92.4	2000

The results in Table 1 indicate that $\varepsilon = 0.10$ offers the best average reward. (1) As for exploration, very small ε (e.g., 0.01) hampers learning due to insufficient exploration. In general, smaller ε values require more training steps to reach optimal performance, while moderate values speed up convergence. (2) Interestingly, the service rate and reward are not strictly aligned. For example, $\varepsilon = 0.15$ yields the highest service rate but the lowest reward, implying that over-fulfilling demand may increase costs. In contrast, $\varepsilon = 0.10$ achieves better overall profit with low service rate. This trade-off is further addressed in Section 2 using a value-based planning approach.

2. Value Iteration Approach

In the given supply chain simulation environment, we identify the following characteristics:

1. The downstream order and target agent's `satisfied_demand` from the previous timestep carry no useful information. Since all non-target agents follow independent random policies (uniformly sampled from $[1, 20]$), any attempt to infer downstream behavior from these signals is inherently unreliable.
2. The environment diverges from realistic supply chain dynamics in a critical way: any order placed to the upstream agent is always fully delivered in the next

Algorithm 1 Value Iteration for Inventory Control

Input: holding cost $h = 0.5$, shortage penalty $c = 2$, unit profit $p = 1$, discount factor $\gamma = 0.99$
Variables:
V_s: value estimate for inventory level s, $s = 1, \ldots, 40$
π_s: optimal order quantity at state s
Initialize V_s with arbitrary values
while not converged **do**
  $V_{\text{copy}} \leftarrow V$ {Copy value function}
  for $s = 1$ **to** 40 **do**
    $v_{\max} \leftarrow -\infty$
    $u \leftarrow \max(\min(40 - s, 20), 1)$ {Upper bound of a}
    for $a = 1$ **to** u **do**
      $sum \leftarrow 0$
      for $d = 1$ **to** 20 **do**
        if $d \leq s$ **then**
          $sum \leftarrow sum + \gamma V_{\text{copy}}[s - d + a] + p \cdot a - h \cdot (s - d + a)$ {Demand fully satisfied}
        else
          $sum \leftarrow sum + \gamma V_{\text{copy}}[a] + p \cdot a - h \cdot a - c \cdot (d - s)$ {$d - s$ demand unsatisfied}
        end if
      end for
      $Q(s, a) \leftarrow sum/20$ {Expected return for (s, a)}
      if $Q(s, a) > v_{\max}$ **then**
        $v_{\max} \leftarrow Q(s, a)$, $a^* \leftarrow a$
      end if
    end for
    $V_s \leftarrow v_{\max}$, $\pi_s \leftarrow a^*$
  end for
end while

timestep. The inventory update follows:

$$\text{inventory}_{t+1} = \text{inventory}_t - \text{satisfied\_demand}_t + \text{orders}_t$$

Even when the upstream agent cannot fulfill the requested amount, only the upstream agent is penalized. The ordering agent still receives the full quantity. We discuss and address this issue in Section 3.

These properties indicate that the optimal policy for the current agent should be independent of both upstream and downstream behaviors. In other words, no useful signal can be extracted from other agents to inform decision-making. As a result, we arrive at a strong conclusion: the optimal action at time t, denoted a_t^*, depends solely on the inventory level at time $t - 1$:

$$a_t^* = \pi^*(\text{inventory}_{t-1}).$$

Let V_s denote the value function when the inventory level is s, corresponding to the expected cumulative return under

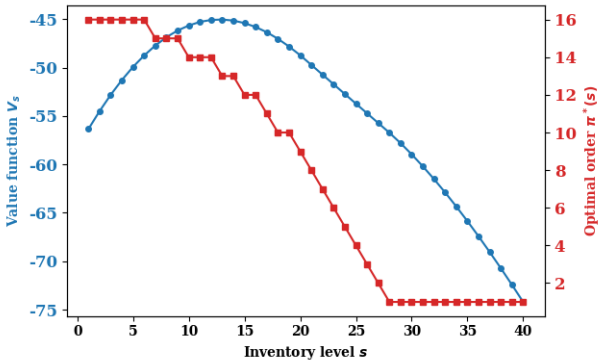

Figure 1. Convergence results of value iteration($\gamma = 0.99$).

the optimal action from that state onward. The Bellman optimality equation (Bellman, 1957) in this context becomes:

$$V_s \leftarrow \max_{a \in \mathcal{A}(s)} \mathbb{E}_{d \sim \mathcal{D}} \left[r(s, a, d) + \gamma V_{s'} \right]$$

where d is the random downstream demand, $r(s, a, d)$ is the immediate reward, and s' is the next inventory level resulting from current inventory, demand, and action.

The detailed algorithm is presented in Algorithm 1. We briefly explain key aspects below:

- **Trade-off in ordering decisions:** The value iteration algorithm balances two opposing objectives: (1) avoiding stockouts, which incur serious penalties due to unsatisfied demand, and (2) minimizing excessive inventory, which leads to holding costs. The optimal order quantity must carefully trade off between these two factors to maximize long-term value.
- **Unit profit term:** The unit profit $p = 1$ represents the difference between the selling price 9 and the purchase cost 8 per unit. Since demand is assumed to be consistently high in the long term, we approximate that all ordered units will eventually be sold. Therefore, the profit $p \cdot a$ is added directly in the value update when an order of size a is placed.
- **Inventory upper bound:** We also exploit a structural insight: any action a such that $s + a > 40$ can be excluded from consideration. A maximum inventory level of 40 units is sufficient to handle worst-case demand (20 units) for the current day, and the agent can replenish inventory as needed in the next timestep. In fact, whenever $s > 40$, the optimal action is always $a^* = 1$, as further ordering only increases holding cost without benefit. Moreover, for any $s \leq 40$, the transition dynamics only involve future inventory levels $s' \leq 40$, ensuring that the state space remains bounded.

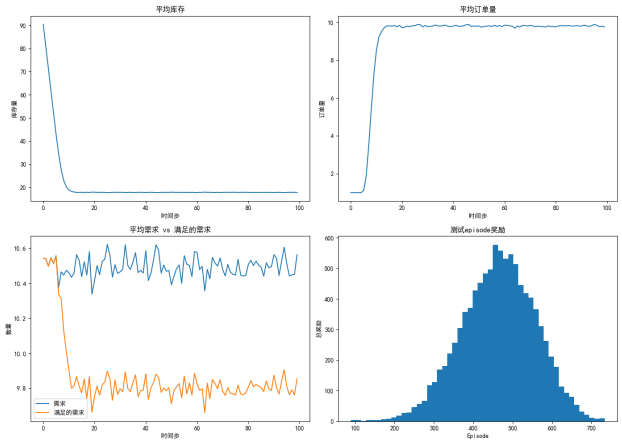

Figure 2. Simulation results of $\pi^*(\gamma = 0.99)$.

This bounded state space simplifies computation and ensures convergence of value iteration.

Value iteration convergence. We adopt a discount factor of $\gamma = 0.99$ in the value iteration procedure, which ensures standard convergence behavior: both the absolute and relative values of V_s stabilize over iterations. This results in a well-defined value function and an optimal policy π^* that balances long-term cost and reward effectively, as guaranteed by classical results on discounted Markov decision processes (Puterman, 1994).

Figure 1 illustrates the converged value function V_s and the corresponding optimal actions $\pi^*(s)$ under the discount factor $\gamma = 0.99$. The value peaks at moderate inventory level $s = 13$ and declines toward both low and high extremes, capturing the trade-off between shortage penalties and holding costs. The learned policy decreases monotonically as inventory increases and eventually stabilizes at the minimal action $a = 1$ when the inventory exceeds 28 units.

The simulation results of the value-based agent are shown in Figure 2. It achieves an average episode reward of **464.10**, improving by 10.27 over the best DQN baseline. The service rate reaches **93.9%**, with an average of 9.86 units of demand fulfilled per step (under random demand with mean 10.5). Behaviorally, the agent quickly reduces its inventory in the first 10 days, then sharply increases its order volume once reaching a low threshold. Afterward, both inventory and ordering stabilize within a dynamic equilibrium range.

Strange convergence with $\gamma = 1$. As an additional experiment, we test $\gamma = 1$, which leads to a different convergence behavior. Specifically, while the relative differences $V_i - V_j$ stabilize over iterations, the absolute values of V_s continue to decrease uniformly due to the negative net reward per timestep. This reflects the fact that, without discounting, the long-run cumulative reward becomes unbounded below in problems with persistent per-step cost. Nevertheless, the learned policy remains similar, yielding comparable performance metrics to the $\gamma = 0.99$ case: average reward **464.22**, service rate **93.8%**.

Furthermore, we find that the **update schedule** of value iteration—whether using in-place updates, forward sweeps, or backward sweeps—has a noticeable impact on convergence speed and the final value estimates. While we empirically observe these effects, we leave a deeper theoretical explanation for future work.

3. Environment Design and Effects

Effect of observation space. In our environment design, the agent is restricted to observing only its local state, specifically the current inventory level. In principle, granting the agent full access to the global environment—such as past demands, upstream actions, or cumulative costs—should not degrade performance, since optimal decisions can still be derived from richer observations. However, as previously analyzed, the agent's optimal action primarily depends on its current inventory. Introducing unnecessary information may increase the complexity of the learning process without providing additional decision value. In practice, we observe that expanding the observation space to full environment actually reduces the reward to **412.43**, likely due to slower convergence and instability in DQN training. This highlights the importance of aligning observation design with the problem's sufficient statistics.

Environment realism enhancements. To better align the simulation environment with real-world supply chain dynamics, we introduce two key modifications:

- **Order fulfillment constraints:** In the default setting, all orders are fulfilled instantly. To introduce realistic supply constraints, we modify the inventory update rule such that downstream inventory depends on upstream availability. Specifically, for agent i, the inventory update becomes:

$$\begin{aligned}\text{inventory}_{i,t+1} = \text{inventory}_{i,t} - \text{satisfied\_demand}_{i,t} \\ + \text{satisfied\_demand}_{i+1,t}\end{aligned}$$

  where $\text{satisfied\_demand}_{i+1,t}$ is the number of items agent $i+1$ can actually deliver, subject to its inventory.

- **Heuristic baseline policy:** We replace the purely random policy with a simple rule-based strategy that maintains a target inventory. Specifically, agent i computes the order quantity a_i at time t as:

$$a_i(t) = \max\left(1, \min\left(20,\ 20 - \left[\text{inventory}_{i,t} - 10\right]\right)\right)$$

where 10 is the expected demand per period and 20 is the target inventory level (twice the expected demand). This rule aims to maintain sufficient stock to meet demand while preventing overordering.

Table 2. Impact of full-environment modeling

Setting	Reward	Service Rate (%)
Without full env	176.86	89.2
With full env	**307.52**	**95.2**

Effect of full-environment modeling. We compare agent performance with and without the full-environment modifications described above. As shown in Table 2, enabling upstream inventory constraints and replacing the random agent with a heuristic baseline leads to significant performance gains: the average reward improves from **176.86** to **307.52**, and the service rate increases from **89.2%** to **95.2%**.

These results highlight the value of incorporating more realistic system dynamics. By modeling upstream supply limits, agents learn to adapt to capacity constraints and better anticipate shortages. Similarly, by observing downstream demand, agents can infer behavioral patterns of their customers and adjust ordering strategies accordingly, leading to more responsive and coordinated supply chain decisions.

References

Bellman, R. *Dynamic Programming*. Princeton University Press, 1957.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Puterman, M. L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.