Table of Contents

Question 1

Linked video provides a demonstration of how robots operate in an Amazon warehouse. The objective is to enable the robot to learn how to navigate through the warehouse and fulfill a list of orders efficiently. By designing an MDP, we can model the decision-making process of the robot in a dynamic environment and help it make optimal decisions to achieve its goals.

It is assumed that the grid map of the warehouse which contains locations of the barriers, the worker and the shelves is known. A sample of the presumed grid map of a warehouse (Figure 1) can be seen below.

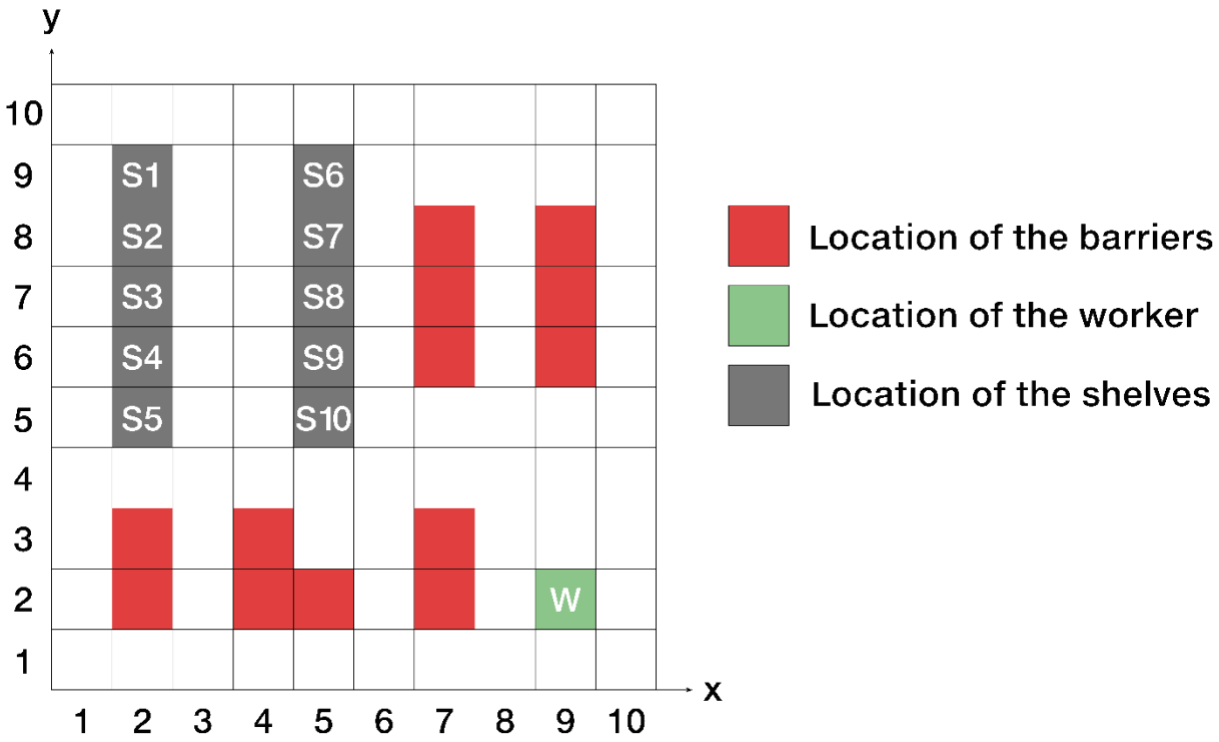

Figure 1 Grid Map of the Warehouse

States of the MDP consist of three parameters: the remaining order list, the current location of the robot and the status of the robot. The robot will start moving the shelves from the top of the given list of orders. List of orders will contain the information of the requested shelf number. The current location of the robot will consist of x and y coordinates. Finally, the status of the robot will consist of the shelf number currently being carried and the current destination of the robot. The current destination of the robot can be listed as the shelf at the top of the remaining list of orders, the worker, being steady and the original location of the shelf. This destination information will be enumerated by 0, 1, 2, 3 respectively. The initial status of the KIVA robot will be the shelf number of the first item of order list and 0 as in the original location of the shelf at the top of the remaining list. After the arrival of the KIVA robot to the worker, there will be a transition from the location of the worker to being steady on the destination of the robot. The being steady section is added to ease the job of the worker take while taking the

product from the shelf. There will be a button on the robot which forces the transition from being steady to going to the original location of the shelf. The initial condition of the robot is (0,0) which implies that there are no shelves being carried and the destination is the location of the shelf at the top of the order list.

States

The states for the MDP of the KIVA robot can be demonstrated as follows:

$$S_t = (O_t, L_t, C_t) = (O_t, (x_t, y_t), (N_t, D_t))$$

$O_t = The\ remaining\ order\ list\ at\ time\ t$
$L_t = \ The\ location\ of\ the\ KIVA\ robot\ at\ time\ t = (x_t, y_t)$
$C_t = The\ condition\ of\ the\ KIVA\ robot\ at\ time\ t = (N_t, D_t)$
$N_t = Shelf\ number\ currently\ being\ carried$

$$D_t = Destination\ of\ the\ robot = \begin{cases} 0 - \ Location\ of\ Shelf\ at\ the\ top\ of\ the\ list \\ 1 - Worker \\ 2 - Being\ Steady \\ 3 - Original\ Location \end{cases}$$

Actions

The action KIVA robot can take can be indicated as follows:

1. Move North: The y coordinate of current position of the KIVA robot increases by 1.
2. Move South: The y coordinate of current position of the KIVA robot decreases by 1.
3. Move East: The x coordinate of current position of the KIVA robot increases by 1.
4. Move West: The x coordinate of current position of the KIVA robot decreases by 1.
5. Pick up: The KIVA robot picks up the shelf from its original location.
6. Drop off: The KIVA robot drops off the shelf being carried at its original location or at the worker's location.
7. Stay: The KIVA robot stays at its current location without taking any action to let the worker get the ordered product.

Rewards

The rewards for the MDP of KIVA robot can be defined as follows:

1. Action: The KIVA robot receives a negative reward of -1 when it takes an action that does not result in any progress towards fulfilling the remaining orders.

2. Collision with a barrier: The KIVA robot receives a large negative reward of -100 when it collides with a barrier or an object in the warehouse. This is to ensure that the robot learns to avoid collisions and navigate safely through the environment.
3. Successful drop-off: The KIVA robot receives a positive reward of +20 when it successfully drops the shelf to its original location.
4. Successful delivery: The KIVA robot receives a positive reward of +20 when the worker gets the ordered shelf.

State Transitions

The state transitions for the MDP in this scenario can be defined as follows:

1. Action Move North: If the KIVA robot is not at the one of the cells whose y coordinate is the largest it can be in the warehouse grid map, and the cell whose y coordinate is 1 point larger than the current position of the robot is not blocked by a barrier, then it can move north. The state transitions to a new state are:

$$S_t' = (O_t, L_t', C_t) = (O_t, (x_t, y_t + 1), C_t)$$

2. Action Move South: If the KIVA robot is not at the one of the cells whose y coordinate is the lowest it can be in the warehouse grid map, and the cell whose y coordinate is 1 point smaller than the current position of the robot is not blocked by a barrier, then it can move south. The state transitions to a new state are:

$$S_t' = (O_t, L_t', C_t) = (O_t, (x_t, y_t - 1), C_t)$$

3. Action Move East: If the KIVA robot is not at the one of the cells whose x coordinate is the largest it can be in the warehouse grid map, and the cell whose x coordinate is 1 point larger than the current position of the robot is not blocked by a barrier, then it can move east. The state transitions to a new state are:

$$S_t' = (O_t, L_t', C_t) = (O_t, (x_t + 1, y_t), C_t)$$

4. Action Move West: If the KIVA robot is not at the one of the cells whose x coordinate is the lowest it can be in the warehouse grid map, and the cell whose x coordinate is 1 point smaller than the current position of the robot is not blocked by a barrier, then it can move west. The state transitions to a new state are:

$$S_t' = (O_t, L_t', C_t) = (O_t, (x_t - 1, y_t), C_t)$$

5. Action Pick-up: If the KIVA robot is at the original shelf location of the top order in the remaining order list, then it can pick up the shelf. Therefore, the condition of the robot will change, and the new destination of the robot will be the location of the worker. The state transitions to a new state:

$$S_t' = (O_t, L_t, C_t') = (O_t, L_t, (N_t, 1))$$

6. Action Drop-off: If the KIVA robot is at the specified drop-off location which is the original location of the shell being carried, then it can drop off the shelf. Thus, the remaining order list and the condition of the robot will be updated. The state transitions to a new state:

$$S_t' = (O_t', L_t, C_t') = (O_t', L_t, (0,0))$$

7. Action Wait: If the KIVA robot is at the worker coordinates, and the button is not being pressed, then it waits. The state remains the same as the previous state.

8. Pressing the button: If the worker presses the button, the resulting state will be the same as the current state, but with the destination set to the original location of the shelf that the robot is currently carrying. The state transitions to a new state:

$$S_t' = (O_t, L_t, C_t') = (O_t, L_t, (L_t, 3))$$

Question 2

Assumptions

- There are m types of items, and n types of customers.
- The lead time is T.
- Customers arrive at the store one at a time.
- When a customer arrives at the store at time t, it is assumed that the customer arrives, decides to buy or not to buy an item, and leaves at time t. In other words, it is assumed that the customers do not spend time in the store, and there is at most one customer in the store at any time t.
- There is a fixed ordering cost if at least one of any items is ordered. Moreover, there is a fixed cost associated with each item, that is, if an item is included in an order, its respective fixed ordering cost should be paid. Finally, there is also the variable cost that the store should pay per item ordered.

Formulating the MDP

States

$$S_t = (D_t, I_t, w_t, r_t)$$

where

$$D_t = (Q_{t-T}, Q_{t-T+1}, Q_{t-T+2}, \dots, Q_{t-1}), \ \ list\ of\ placed\ orders\ from\ T-t\ until\ t$$

$$Q_t = \left(Q_{1,t}, Q_{2,t}, \dots, Q_{m,t}\right), \ \ placed\ order\ at\ time\ t$$

$$Q_{i,t} = placed\ order\ quantity\ of\ item\ i\ at\ time\ t, \ \ i\epsilon(1,2,\dots,m)$$

$$I_t = \left(I_{1,t}, I_{2,t}, \dots, I_{m,t}\right), \ \ list\ of\ inventory\ levels\ of\ items\ at\ time\ t$$

$$I_{i,t} = inventory\ level\ of\ item\ i\ at\ time\ t, \ \ i\epsilon(1,2,\dots,m)$$

$$w_t = \begin{cases} 0, & it\ is\ not\ possible\ to\ order\ new\ items\ at\ time\ t \\ 1, & it\ is\ possible\ to\ order\ new\ items\ at\ time\ t \end{cases}$$

$$r_t = \begin{cases} 0, & there\ is\ no\ customer\ at\ the\ store\ at\ time\ t \\ j, & there\ is\ a\ customer\ of\ type\ j\ at\ the\ store\ at\ time\ t, \ \ j\epsilon(1,2,\dots,n) \end{cases}$$

Actions

$$A_t = (Q_t, Y_t)$$

where

$$Q_t = (Q_{1,t}, Q_{2,t}, \dots, Q_{m,t})$$

$$Q_{i,t} = placed\ order\ quantity\ of\ item\ i\ at\ time\ t,\ \ i\epsilon(1,2,\dots,m)$$

$$Y_t = (Y_{1,t}, Y_{2,t}, \dots, Y_{m,t})$$

$$Y_{i,t} = \begin{cases} 0, & item\ i\ is\ not\ displayed\ at\ time\ t,\ \ i\epsilon(1,2,\dots,m) \\ 1, & item\ i\ is\ displayed\ at\ time\ t,\ \ i\epsilon(1,2,\dots,m) \end{cases}$$

Note that if $w_t = 0$, then $Q_{i,t} = 0\ \forall\ i\epsilon(1,2,\dots,m)$

Note that if $I_{i,t} = 0$, then $Y_{i,t} = 0\ \forall\ i\epsilon(1,2,\dots,m)$

State Transitions

$$Z_{i,t} = \begin{cases} 0, & item\ i\ is\ not\ sold\ at\ time\ t \\ 1, & item\ i\ is\ sold\ at\ time\ t \end{cases}$$

Note that if $Y_{i,t} = 0$, then $Z_{i,t} = 0\ \forall\ i\epsilon(1,2,\dots,m)$

The inventory level component of the next state can be expressed as below:

$$I_{i,t+1} = I_{i,t} - Z_{i,t} + Q_{i,t-T}$$

where $Z_{i,t}$ depends on the realized utility level of the customer, and the display status of items.

The other components of the next state:

$$D_{t+1} = (Q_{t-T+1}, Q_{t-T+2}, Q_{t-T+3}, \dots, Q_t)$$

$$w_t = \begin{cases} 0, & it\ is\ not\ possible\ to\ order\ new\ items\ at\ time\ t \\ 1, & it\ is\ possible\ to\ order\ new\ items\ at\ time\ t \end{cases}$$

$$r_t = \begin{cases} 0, & there\ is\ no\ customer\ at\ the\ store\ at\ time\ t \\ j, & there\ is\ a\ customer\ of\ type\ j\ at\ the\ store\ at\ time\ t,\ \ j\epsilon(1,2,\dots,n) \end{cases}$$

where w_t depends on the time of the day, and r_t depends on a Poisson distribution.

Rewards

Let p_i be the profit from selling an item i where $i\epsilon(1,2,\dots,m)$.

Let K be the fixed ordering cost.

Let X_i be the fixed cost of ordering item i where $i\epsilon(1,2,\dots,m)$.

Let c_i be the variable ordering cost per item i where $i\epsilon(1,2,\dots,m)$.

Let $F_{i,t} = \begin{cases} 0, & no\ item\ i\ is\ ordered\ at\ time\ t \\ 1, & at\ least\ 1\ item\ i\ is\ ordered\ at\ time\ t \end{cases}$ $,i\epsilon(1,2,\dots,m)$

Let $F_t = \begin{cases} 0, & \sum_{i=1}^{m} F_{i,t} = 0 \\ 1, & \sum_{i=1}^{m} F_{i,t} > 0 \end{cases}$

The reward after taking an action at any time t can be expressed as:

$$R_t = \sum_{i=1}^{m} Z_{i,t} * p_i - K * F_t - \sum_{i=1}^{m} X_i * F_{i,t} - \sum_{i=1}^{m} c_i * Q_{i,t}$$

Question 3

Ways of Simplification

While formulating the MDP for Question 2, it has been observed that the uncertainty in the utility that a particular customer type will receive from a particular item creates difficulties. To avoid this, it can be assumed that customers of the same types will derive equal utility from the same type of items. In other words, eliminating the uncertainty factor (ϵ_j) from the original utility function ($u_{ij} = v_{ij} + \epsilon_j$) makes the utility only depend on the item, customer type pair. With this change, the priority items of all types of customers can be detected easily by just checking the fixed v'_{ij} values. When a customer comes to the store, it can be known surely which item will be bought by the customer according to the displayed items. In this situation, some simplifications can be made in action-value functions.

Secondly, to simplify the order process, it is suggested that instead of constantly changing which items to display for different customer types, this decision can be made at the start of each day when placing the orders. This means that the store decides which items to display for each customer type at the start of each day. Hence, only one action is required per day for orders and display decisions. This simplification provides a remarkable reduction in both state and action spaces. With this change, w_t and r_t components of the states become redundant, and they can be removed from the MDP. Also, the time parameter t is defined as one day.

Reformulating the MDP

To make settings clearer, the lead time T is set to 2, the number of customer types n is set to 3, and the number of item types m is set to 5.

States

$$S_t = (D_t, I_t)$$

where

$$D_t = (Q_{t-2}, Q_{t-1}),\ \ list\ of\ placed\ orders\ from\ day\ t-2\ until\ day\ t$$

$$Q_t = \left(Q_{1,t}, Q_{2,t}, \dots, Q_{5,t}\right),\ \ placed\ order\ at\ the\ start\ of\ day\ t$$

$$Q_{i,t} = placed\ order\ quantity\ of\ item\ i\ at\ the\ start\ of\ day\ t,\ \ i\epsilon(1,2,\dots,5)$$

$$I_t = \left(I_{1,t}, I_{2,t}, \dots, I_{5,t}\right), \ \ list\ of\ inventory\ levels\ of\ items\ at\ the\ start\ of\ day\ t$$

$$I_{i,t} = inventory\ level\ of\ item\ i\ at\ the\ start\ of\ day\ t, \ \ i\epsilon(1,2,\dots,5)$$

Actions

$$A_t = (Q_t, H_t)$$

where

$$Q_t = (Q_{1,t}, Q_{2,t}, \dots, Q_{5,t})$$

$$Q_{i,t} = placed\ order\ quantity\ of\ item\ i\ at\ the\ start\ of\ day\ t, \ \ i\epsilon(1,2,\dots,5)$$

$$H_t = (H_{1,t}, H_{2,t}, H_{3,t})$$

$$H_{j,t} = (h_{1,j,t}, h_{2,j,t}, \dots, h_{5,j,t}), \ \ j\epsilon(1,2,3)$$

$$h_{i,j,t} = \begin{cases} 0, item\ i\ is\ not\ displayed\ for\ customer\ type\ j\ in\ day\ t \\ 1, \quad item\ i\ is\ displayed\ for\ customer\ type\ j\ in\ day\ t \end{cases} \quad , i\epsilon(1,2,\dots,5), j\epsilon(1,2,3)$$

Note that if $I_{i,t} = 0$, then $h_{i,j,t} = 0\ \forall\ i\epsilon(1,2,\dots,5), j\epsilon(1,2,3)$

State Transitions

$$Z_{i,t} = number\ of\ item\ i\ sold\ in\ day\ t$$

Note that $Z_{i,t} \leq I_{i,t}\ \forall\ i\epsilon(1,2,\dots,5)$

The inventory level component of the next state can be expressed as below:

$$I_{i,t+1} = I_{i,t} - Z_{i,t} + Q_{i,t-2}$$

where $Z_{i,t}$ depends on the realized utility level of customers, and the display status of items.

The other component of the next state:

$$D_{t+1} = (Q_{t-1}, Q_t)$$

Rewards

Let p_i be the profit from selling an item i where $i\epsilon(1,2,\dots,5)$.

Let K be the fixed ordering cost.

Let X_i be the fixed cost of ordering item i where $i\epsilon(1,2,\dots,5)$.

Let c_i be the variable ordering cost per item i where $i\epsilon(1,2,\dots,5)$.

Let $F_{i,t} = \begin{cases} 0, & no\ item\ i\ is\ ordered\ at\ the\ start\ of\ day\ t \\ 1, & at\ least\ 1\ item\ i\ is\ ordered\ at\ the\ start\ of\ day\ t \end{cases}$, $i\epsilon(1,2,\dots,5)$

Let $F_t = \begin{cases} 0, & \sum_{i=1}^{5} F_{i,t} = 0 \\ 1, & \sum_{i=1}^{5} F_{i,t} > 0 \end{cases}$

The reward after taking an action in any day t can be expressed as:

$$R_t = \sum_{i=1}^{5} Z_{i,t} * p_i - K * F_t - \sum_{i=1}^{5} X_i * F_{i,t} - \sum_{i=1}^{5} c_i * Q_{i,t}$$

Question 4

4a.

Formulating the MDP

States

$$S_t = (I_t, O_t)$$

$I_t = \ Inventory\ level\ at\ time\ t$
$O_t = the\ order\ made\ at\ the\ beginning\ of\ previous\ period$

Actions

$$A_t\ \epsilon\ (0, 20, 50), the\ order\ amounts\ possible\ at\ time\ t$$

State Transitions

$$Demand\ D_t = \begin{cases} 10, & p = 0.25 \\ 30, & p = 0.75 \end{cases}$$

Therefore, the next state can be expressed as below:

$$S_{t+1} = (min(50, max(0, I_t - D_t) + O_t),\ A_t)$$

where the demand term D_t follows a discrete probability distribution as given above.

Rewards

The immediate reward after taking an action at any period t can be expressed as:

$$Reward\ From\ Action\ = \begin{cases} 0, & A_t = 0 \\ -24, & A_t = 20 \\ -50, & A_t = 50 \end{cases}$$

$$R_t = Reward\ From\ Action\ + max(0, I_t - D_t) \times -1\ + min(0, I_t - D_t) \times 20$$

State-Value Function

Let $r = R_t,\ s = S_t,\ a = A_t,\ s' = S_{t+1}$:

$$V_\pi(s) = E_\pi[G_t \mid S_t = s] = E_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] =$$

$$= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \mid s, a)[r\ +\ \gamma E_\pi[G_{t+1} \mid S_{t+1} = s']]$$

$$= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \mid s, a)[r + \gamma V_\pi(s')]$$

Assuming a deterministic policy, the state-value function is,

$$\boldsymbol{V_\pi(s) = 0.25 \times [r_1 + \gamma V_\pi(s'_1)] + 0.75 \times [r_2 + \gamma V_\pi(s'_2)]}$$

where

$$r_1 = Reward\ From\ Action + \max(0, I_t - 10) \times -1 + \min(0, I_t - 10) \times 20$$

$$r_2 = Reward\ From\ Action + max(0, I_t - 30) \times -1 + min(0, I_t - 30) \times 20$$

$$s'_1 = (min(50, max(0, I_t - 10) + O_t), a)$$

$$s'_2 = (min(50, max(0, I_t - 30) + O_t), a)$$

$$Reward\ From\ Action = \begin{cases} 0, & a = 0 \\ -24, & a = 20 \\ -50, & a = 50 \end{cases}$$

Optimality Equation for State-Value Function

$$V_*(s) = max_\pi V_\pi(s) = max_a E[R_{t+1} + \gamma V_*(S_{t+1}) | S_t = s, A_t = a]$$

$$= max_a \sum_{s'} \sum_r p(s', r \mid s, a)[r + \gamma V_*(s')]$$

$$\boldsymbol{= max_a(0.25 \times [r_1 + \gamma V_*(s'_1)] + 0.75 \times [r_2 + \gamma V_*(s'_2)])}$$

where

$$r_1 = Reward\ From\ Action + \max(0, I_t - 10) \times -1 + \min(0, I_t - 10) \times 20$$

$$r_2 = Reward\ From\ Action + max(0, I_t - 30) \times -1 + min(0, I_t - 30) \times 20$$

$$s'_1 = (min(50, max(0, I_t - 10) + O_t), a)$$

$$s'_2 = (min(50, max(0, I_t - 30) + O_t), a)$$

$$Reward\ From\ Action = \begin{cases} 0, & a = 0 \\ -24, & a = 20 \\ -50, & a = 50 \end{cases}$$

4b.

Iterative policy evaluation

In this part of the assignment, we will evaluate the initial policy. After initializing the state-value function to a zeros array, we will loop through all states and compute new state values using dynamic programming.

State-value function equation from the previous part will be used to make the calculations.

Assuming $\gamma = 0.8$.

$$\boldsymbol{V_\pi(s) = 0.25 \times [r_1 + \gamma V_\pi(s'_1)] + 0.75 \times [r_2 + \gamma V_\pi(s'_2)]}$$

$Policy\ \pi\ to\ be\ evaluated: Order\ when\ I_t + O_t = 0.$

State	Action	State	Action	State	Action
(0, 0)	50	(0, 20)	0	(0, 50)	0
(10, 0)	0	(10, 20)	0	(10, 50)	0
(20, 0)	0	(20, 20)	0	(20, 50)	0
(30, 0)	0	(30, 20)	0	(30, 50)	0
(40, 0)	0	(40, 20)	0	(40, 50)	0
(50, 0)	0	(50, 20)	0	(50, 50)	0

Policy to be evaluated

$Initialization: V(s) = 0\ for\ all\ states$

<u>1st Iteration</u>

Looping through all states:

$\boldsymbol{s = (0, 0)}$

$a = order\ 50$

$demand = 10 => r_1 = -50 + -10 \times 20 = -250$

$demand = 30 => r_2 = -50 + -30 \times 20 = -650$

$s'_1 = s'_2 = (0, 0) => V_\pi(s'_1) = V_\pi(s'_2) = 0$

$V_\pi((0,0)) = 0.25 \times [-250 + 0.8 \times 0] + 0.75 \times [-650 + 0.8 \times 0] = -550$

$\boldsymbol{s = (0, 20)}$

$a = order\ 0$

$demand = 10 => r_1 = 0 + -10 \times 20 = -200$

$demand = 30 => r_2 = 0 + -30 \times 20 = -600$

$s_1' = s_2' = (20, 0) => V_\pi(s_1') = V_\pi(s_2') = 0$

$V_\pi((0,20)) = 0.25 \times [-200 + 0.8 \times 0\ ] + 0.75 \times [-600 + 0.8 \times 0\ ] = -500$

$\boldsymbol{s = (0, 50)}$

$a = order\ 0$

$demand = 10 => r_1 = 0 + -10 \times 20 = -200$

$demand = 30 => r_2 = 0 + -30 \times 20 = -600$

$s_1' = s_2' = (20, 0) => V_\pi(s_1') = V_\pi(s_2') = 0$

$V_\pi((0,50)) = 0.25 \times [-200 + 0.8 \times 0\ ] + 0.75 \times [-600 + 0.8 \times 0\ ] = -500$

State	Value	State	Value	State	Value
(0, 0)	0.0	(0, 20)	0.0	(0, 50)	0.0
(10, 0)	0.0	(10, 20)	0.0	(10, 50)	0.0
(20, 0)	0.0	(20, 20)	0.0	(20, 50)	0.0
(30, 0)	0.0	(30, 20)	0.0	(30, 50)	0.0
(40, 0)	0.0	(40, 20)	0.0	(40, 50)	0.0
(50, 0)	0.0	(50, 20)	0.0	(50, 50)	0.0

From now on, we will not show the detailed calculations to make the iterations look clearer and more understandable.

$\boldsymbol{s = (10, 0)}$ $=>$ $V_\pi((10,0)) = 0.25 \times [0 + 0.8 \times -550\ ] + 0.75 \times [-400 + 0.8 \times -550\ ] = -740$

$\boldsymbol{s = (10, 20)}$ $=>$ $V_\pi((10,20)) = 0.25 \times [0 + 0.8 \times 0\ ] + 0.75 \times [-400 + 0.8 \times 0\ ] = -300$

$\boldsymbol{s = (10, 50)}$ $=>$ $V_\pi((10,50)) = 0.25 \times [0 + 0.8 \times 0\ ] + 0.75 \times [-400 + 0.8 \times 0\ ] = -300$

$\boldsymbol{s = (20, 0)}$ $=>$ $V_\pi((20,0)) = 0.25 \times [-10 + 0.8 \times -740\ ] + 0.75 \times [-200 + 0.8 \times -550\ ] = -630.5$

$\boldsymbol{s = (20, 20)}$ $=>$ $V_\pi((20,20)) = 0.25 \times [-10 + 0.8 \times 0\ ] + 0.75 \times [-200 + 0.8 \times -630.5\ ] = -530.8$

$\boldsymbol{s = (20, 50)}$ $=>$ $V_\pi((20,50)) = 0.25 \times [-10 + 0.8 \times 0\ ] + 0.75 \times [-200 + 0.8 \times 0\ ] = -152.5$

$\boldsymbol{s} = (\mathbf{30}, \mathbf{0}) \Rightarrow V_\pi((30,0)) = 0.25 \times [-20 + 0.8 \times -630.5\ ] + 0.75 \times [0 + 0.8 \times -550\ ] = -461.1$

$\boldsymbol{s} = (\mathbf{30}, \mathbf{20}) \Rightarrow V_\pi((30,20)) = 0.25 \times [-20 + 0.8 \times 0\ ] + 0.75 \times [0 + 0.8 \times -630.5\ ] = -383.3$

$\boldsymbol{s} = (\mathbf{30}, \mathbf{50}) \Rightarrow V_\pi((30,50)) = 0.25 \times [-20 + 0.8 \times 0\ ] + 0.75 \times [0 + 0.8 \times 0\ ] = -5$

$\boldsymbol{s} = (\mathbf{40}, \mathbf{0}) \Rightarrow V_\pi((40,0)) = 0.25 \times [-30 + 0.8 \times -461.1\ ] + 0.75 \times [-10 + 0.8 \times -740\ ] = -551.2$

$\boldsymbol{s} = (\mathbf{40}, \mathbf{20}) \Rightarrow V_\pi((40,20)) = 0.25 \times [-30 + 0.8 \times 0\ ] + 0.75 \times [-10 + 0.8 \times -461.1\ ] = -291.7$

$\boldsymbol{s} = (\mathbf{40}, \mathbf{50}) \Rightarrow V_\pi((40,50)) = 0.25 \times [-30 + 0.8 \times 0\ ] + 0.75 \times [-10 + 0.8 \times 0\ ] = -15$

$\boldsymbol{s} = (\mathbf{50}, \mathbf{0}) \Rightarrow V_\pi((50,0)) = 0.25 \times [-40 + 0.8 \times -551.2\ ] + 0.75 \times [-20 + 0.8 \times -630.5\ ] = -513.5$

$\boldsymbol{s} = (\mathbf{50}, \mathbf{20}) \Rightarrow V_\pi((50,20)) = 0.25 \times [-40 + 0.8 \times -513.5\ ] + 0.75 \times [-20 + 0.8 \times -551.2\ ] = -458.4$

$\boldsymbol{s} = (\mathbf{50}, \mathbf{50}) \Rightarrow V_\pi((50,50)) = 0.25 \times [-40 + 0.8 \times -513.5\ ] + 0.75 \times [-20 + 0.8 \times -513.5\ ] = -435.8$

State values after 1st iteration of policy evaluation algorithm:

State	Value	State	Value	State	Value
(0, 0)	-550.0	(0, 20)	-500.0	(0, 50)	-500.0
(10, 0)	-740.0	(10, 20)	-300.0	(10, 50)	-300.0
(20, 0)	-630.5	(20, 20)	-530.8	(20, 50)	-152.5
(30, 0)	-461.1	(30, 20)	-383.3	(30, 50)	-5.0
(40, 0)	-551.2	(40, 20)	-291.7	(40, 50)	-15.0
(50, 0)	-513.5	(50, 20)	-458.4	(50, 50)	435.8

2nd Iteration

$\boldsymbol{s} = (\mathbf{0}, \mathbf{0}) \Rightarrow V_\pi((0,0)) = 0.25 \times [-250 + 0.8 \times -500\ ] + 0.75 \times [-650 + 0.8 \times -500\ ] = -950$

$\boldsymbol{s} = (\mathbf{0}, \mathbf{20}) \Rightarrow V_\pi((0,20)) = 0.25 \times [-200 + 0.8 \times -630.5\ ] + 0.75 \times [-600 + 0.8 \times -630.5\ ] = -1004.4$

$\boldsymbol{s} = (\mathbf{0}, \mathbf{50}) \Rightarrow V_\pi((0,50)) = 0.25 \times [-200 + 0.8 \times -513.5\ ] + 0.75 \times [-600 + 0.8 \times -513.5\ ] = -910.8$

$\boldsymbol{s} = (\mathbf{10}, \mathbf{0}) \Rightarrow V_\pi((10,0)) = 0.25 \times [0 + 0.8 \times -950\ ] + 0.75 \times [-400 + 0.8 \times -950\ ] = -1060$

$\boldsymbol{s} = (\mathbf{10}, \mathbf{20}) \Rightarrow V_\pi((10,20)) = 0.25 \times [0 + 0.8 \times -630.5\ ] + 0.75 \times [-400 + 0.8 \times -630.5\ ] = -804.4$

$\boldsymbol{s} = (\mathbf{10}, \mathbf{50}) \Rightarrow V_\pi((10,50)) = 0.25 \times [0 + 0.8 \times -513.5\ ] + 0.75 \times [-400 + 0.8 \times -513.5\ ] = -710.8$

$\boldsymbol{s} = (\mathbf{20}, \mathbf{0}) \Rightarrow V_\pi((20,0)) = 0.25 \times [-10 + 0.8 \times -1060\ ] + 0.75 \times [-200 + 0.8 \times -950\ ] = -934.5$

$\boldsymbol{s} = (\mathbf{20}, \mathbf{20}) \Rightarrow V_\pi((20,20)) = 0.25 \times [-10 + 0.8 \times -461.1\ ] + 0.75 \times [-200 + 0.8 \times -934.5\ ] = -805.4$

$\boldsymbol{s} = (\mathbf{20}, \mathbf{50}) \Rightarrow V_\pi((20,50)) = 0.25 \times [-10 + 0.8 \times -513.5\ ] + 0.75 \times [-200 + 0.8 \times -513.5\ ] = -563.3$

$\boldsymbol{s} = (\mathbf{30}, \mathbf{0}) \Rightarrow V_\pi((30,0)) = 0.25 \times [-20 + 0.8 \times -934.5\ ] + 0.75 \times [0 + 0.8 \times -950\ ] = -761.9$

$\boldsymbol{s} = (\mathbf{30}, \mathbf{20}) \Rightarrow V_\pi((30,20)) = 0.25 \times [-20 + 0.8 \times -551.2\ ] + 0.75 \times [0 + 0.8 \times -934.5\ ] = -675.9$

$\boldsymbol{s} = (\mathbf{30}, \mathbf{50}) \Rightarrow V_\pi((30,50)) = 0.25 \times [-20 + 0.8 \times -513.5\ ] + 0.75 \times [0 + 0.8 \times -513.5\ ] = -415.8$

$\boldsymbol{s} = (\mathbf{40}, \mathbf{0}) \Rightarrow V_\pi((40,0)) = 0.25 \times [-30 + 0.8 \times -761.9\ ] + 0.75 \times [-10 + 0.8 \times -1060\ ] = -803.4$

$\boldsymbol{s} = (\mathbf{40}, \mathbf{20}) \Rightarrow V_\pi((40,20)) = 0.25 \times [-30 + 0.8 \times -513.5\ ] + 0.75 \times [-10 + 0.8 \times -761.9\ ] = -574.8$

$\boldsymbol{s} = (\mathbf{40}, \mathbf{50}) \Rightarrow V_\pi((40,50)) = 0.25 \times [-30 + 0.8 \times -513.5\ ] + 0.75 \times [-10 + 0.8 \times 0 - 513.5] = -425.8$

$\boldsymbol{s} = (\mathbf{50}, \mathbf{0}) \Rightarrow V_\pi((50,0)) = 0.25 \times [-40 + 0.8 \times -803.4\ ] + 0.75 \times [-20 + 0.8 \times -934.5\ ] = -746.4$

$\boldsymbol{s} = (\mathbf{50}, \mathbf{20}) \Rightarrow V_\pi((50,20)) = 0.25 \times [-40 + 0.8 \times -746.4\ ] + 0.75 \times [-20 + 0.8 \times -803.4\ ] = -656.3$

$\boldsymbol{s} = (\mathbf{50}, \mathbf{50}) \Rightarrow V_\pi((50,50)) = 0.25 \times [-40 + 0.8 \times -746.4\ ] + 0.75 \times [-20 + 0.8 \times -746.4] = -435.8$

State values after 2nd iteration of policy evaluation algorithm:

State	Value	State	Value	State	Value
(0, 0)	-950.0	(0, 20)	-1004.4	(0, 50)	-910.8
(10, 0)	-1060.0	(10, 20)	-804.4	(10, 50)	-710.8
(20, 0)	-934.5	(20, 20)	-805.4	(20, 50)	-563.3
(30, 0)	-761.9	(30, 20)	-675.9	(30, 50)	-415.8
(40, 0)	-803.4	(40, 20)	-574.8	(40, 50)	-425.8
(50, 0)	-746.4	(50, 20)	-656.3	(50, 50)	-622.1

4c.

Policy Improvement Algorithm

Initial policy was:

State	Action	State	Action	State	Action
(0, 0)	50	(0, 20)	0	(0, 50)	0
(10, 0)	0	(10, 20)	0	(10, 50)	0
(20, 0)	0	(20, 20)	0	(20, 50)	0
(30, 0)	0	(30, 20)	0	(30, 50)	0
(40, 0)	0	(40, 20)	0	(40, 50)	0
(50, 0)	0	(50, 20)	0	(50, 50)	0

We will loop through each state, get the action that previous policy gives, then compare it with the action that maximizes the reward. If the actions are different, we will update the policy for the corresponding state.

Starting policy improvement step.

$\boldsymbol{s} = (\mathbf{0}, \mathbf{0})$

$old\ action = order\ 50$

$demand = 10 => r_1 = -50 + -10 \times 20 = -250$

$demand = 30 => r_2 = -50 + -30 \times 20 = -650$

$s_1' = s_2' = (0,0) => V_\pi(s_1') = V_\pi(s_2') = 0$

$$\pi(s) = argmax_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')]$$

$$= argmax_a \begin{cases} 0.25 \times [-200 + 0.8 \times -950\ ] + 0.75 \times [-600 + 0.8 \times -950\ ] = -1260, & a = 0 \\ 0.25 \times [-224 + 0.8 \times -1004.4\ ] + 0.75 \times [-624 + 0.8 \times -1004.4\ ] = -1327.5, & a = 20 \\ 0.25 \times [-250 + 0.8 \times -910.8\ ] + 0.75 \times [-650 + 0.8 \times -910.8\ ] = -1278.6, & a = 50 \end{cases}$$

$= 50\ (optimal\ action\ is\ to\ order\ 50)$

$\boldsymbol{s} = (\mathbf{0}, \mathbf{20})$

$old\ action = order\ 0$

$demand = 10 => r_1 = -50 + -10 \times 20 = -250$

$demand = 30 => r_2 = -50 + -30 \times 20 = -650$

$s_1' = s_2' = (0,0) => V_\pi(s_1') = V_\pi(s_2') = 0$

$$\pi(s) = argmax_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')]$$

$$= argmax_a \begin{cases} 0.25 \times [-200 + 0.8 \times -934.5\ ] + 0.75 \times [-600 + 0.8 \times -934.5\ ] = -1247.6, & a = 0 \\ 0.25 \times [-224 + 0.8 \times -805.4\ ] + 0.75 \times [-624 + 0.8 \times -805.4\ ] = -1168.3, & a = 20 \\ 0.25 \times [-250 + 0.8 \times -563.3\ ] + 0.75 \times [-650 + 0.8 \times -563.3] = -1000.6, & a = 50 \end{cases}$$

$= 50\ (optimal\ action\ is\ to\ order\ 50)$

From now on, we will not give detailed calculations but only the results.

$$\boldsymbol{s} = (\mathbf{0}, \mathbf{50}) => \pi(s) = argmax_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')] = 0$$

$$\boldsymbol{s = (10,0)} => \pi(s) = argmax_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')] = 0$$

$$\boldsymbol{s = (10,20)} => \pi(s) = argmax_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')] = 50$$

$$\boldsymbol{s = (10,50)} => \pi(s) = argmax_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')] = 0$$

$$\boldsymbol{s = (20,0)} => \pi(s) = argmax_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')] = 50$$

$$\boldsymbol{s = (20,20)} => \pi(s) = argmax_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')] = 50$$

$$\boldsymbol{s = (20,50)} => \pi(s) = argmax_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')] = 0$$

$$\boldsymbol{s = (30,0)} => \pi(s) = argmax_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')] = 50$$

$$\boldsymbol{s = (30,20)} => \pi(s) = argmax_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')] = 50$$

$$\boldsymbol{s = (30,50)} => \pi(s) = argmax_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')] = 0$$

$$\boldsymbol{s = (40,0)} => \pi(s) = argmax_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')] = 50$$

$$\boldsymbol{s = (40,20)} => \pi(s) = argmax_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')] = 50$$

$$\boldsymbol{s = (40,50)} => \pi(s) = argmax_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')] = 0$$

$$\boldsymbol{s = (50,0)} => \pi(s) = argmax_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')] = 50$$

$$\boldsymbol{s = (50,20)} => \pi(s) = argmax_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')] = 50$$

$$\boldsymbol{s = (50,50)} => \pi(s) = argmax_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')] = 50$$

Therefore, the new policy is as below:

State	Action	State	Action	State	Action
(0, 0)	50	(0, 20)	50	(0, 50)	0
(10, 0)	0	(10, 20)	50	(10, 50)	0
(20, 0)	50	(20, 20)	50	(20, 50)	0
(30, 0)	50	(30, 20)	50	(30, 50)	0
(40, 0)	50	(40, 20)	50	(40, 50)	0
(50, 0)	50	(50, 20)	50	(50, 50)	50

Since the new policy we obtained was different from the initial policy, we can conclude that the initial policy was not optimal.

4d.

In policy iteration, we start with a random policy and run two separate algorithms until convergence: policy evaluation and improvement. We run policy evaluation until convergence and then update the policy by a policy improvement step. Policy iteration stops when there are no more improvements to the policy in the policy improvement step. In value iteration, however; we do not run the policy evaluation step until convergence but rather make one iteration and then take the action that has the maximum value and continue until state values converge to a value (i.e. each $v \in V(s) < \theta \; where \; \theta \; is \; a \; small \; value \; of \; our \; choice$). Then, the optimal policy is calculated once the state values are converged.

Question 5

You can refer to the "1CM240_q5.ipynb" file submitted for the solution of this part. We have also submitted the html version of this file. Before running the ipynb file, we recommend putting the below files in the same directory as the ipynb file. If they are not in the same directory, they should be trained for a very long time, 15 minutes for policy evaluation step, 100 minutes for policy improvement step, and 10 hours (15 million iterations) for Q-learning step. Below are the files for this question:

- *"1CM240_q5.ipynb"*: Original ipynb file
- *"1CM240_q5.html"*: Html copy of the original ipynb file
- *"initial_policy_V.pkl"*: State values for the base policy after policy evaluation step
- *"policy_after_improvement.pkl"*: Policy obtained after one step of policy improvement algorithm
- *"q_dict.pkl"*: Original q table (already trained by us)

Summary and remarks of the results of Question 5

In question 5, we started with evaluating a base policy. After implementing the policy evaluation and improvement steps, we concluded that the base policy was not optimal. Since the runtime for the policy iteration algorithm was too high (approx. 2 hours for each iteration) we used another approach, q-learning algorithm to deal with the problem. We trained our Q-learning algorithm for more than 10 million episodes, each episode consisting of a 30-day simulation with random demands from the gamma distribution. We started with a discount factor of 0.9 and decreased it throughout the training process. Our observations were as below after we were done with training and observed the results.

In Q5.D in the code, you can see a sample episode, starting with a random inventory and ran for 30 days. It is clearly visible that the route option (1,2,3) was never used. This is because of the high cost (-500) associated with this route. Our algorithm learned that it is better to bear the sales loss than trying to supply to all three of the stores at the same time. The algorithm achieved an average immediate reward of -67.9 for a typical day, which can be considered as very good.

Our algorithm also learned that facing a possible sales loss is much worse than paying for excess inventory. We can see that it preferred to order more than the expected mean demands most of the time. This is also partly because of the fact that all three of the stores cannot be supplied at the same time because of the high cost associated with it. Therefore, it learned that an excess inventory should almost always be available because of this reason. In a 30-day simulation of the system below you can see that the order-up-to inventories are relatively higher than the mean of the gamma distributions as a result.

Question 6

You can refer to the "1CM240_q6.ipynb" file submitted for the solution of this part. We have also submitted the html version of this file. Before running the ipynb file, we recommend putting our pre-trained q learning models in the same directory as the ipynb file. If they are not in the same directory, they should be trained for 1.5 million episodes from scratch to produce the same results (training will take around 1.5 hours for each). Below are the files for this question:

- *"1CM240_q6.ipynb"*: Original ipynb file
- *"1CM240_q6.html"*: Html copy of the original ipynb file
- *"q_dict_q6.pkl"*: First q-table (already trained by us)
- *"q_dict_cost1.pkl"*: Second q-table (already trained by us)
- *"q_dict_cost2.pkl"*: Third q-table (already trained by us)

Summary of the results of Question 6

In this question, we tried to come up with a faster and better solution for the logistics problem we had. While training both policy iteration and tabular q-learning algorithm in question 5, we struggled with the problem of very long training times and curses of dimensionality because of the big state and action space we had. To overcome this issue, we made use of the observations we made from our nearly optimal q-learning model we trained in question 5. First and most obvious observation that we made for this question was that the route (1,2,3) is very costly, and it should never be used since its cost is much higher than just bearing loss sales. Therefore, we excluded this route from the problem in this part. By doing this, we reduced action space from having 1331 elements (all combination of 3 integers between 0-10) to 331 elements (by subtracting all combinations of 3 integers between 1-10).

We also carried out a sensitive analysis to see how different cost values affects the decisions of our q-learning algorithm makes. To achieve this, we trained 3 different versions of this algorithm:

- 1st model: Original Version (excluding the route (1,2,3))
- 2nd model: Original Version with holding cost being -10 instead of -1
- 3rd model: Original version with higher route costs for routes (1,2), (1,3), (2,3)

All the below results are also available in the ipynb file and can be regenerated by simply running the cells after putting the files we mentioned above in the same directory. Below are the results for the mean order-up-to levels (inventory levels after ordering):

Models		
1st model	**2nd model**	**3rd model**

Store 1	Store 2	Store 3	Store 1	Store 2	Store 3	Store 1	Store 2	Store 3
6.95	8.16	7.51	4.54	6.22	2.74	5.41	6.11	5.39

Result 1.

Route Frequencies			
	1st Model	**2nd Model**	**3rd Model**
No supply	10	3	3
(1)	7	10	31
(2)	27	18	71
(3)	4	5	26
(1,2)	140	154	77
(1,3)	11	19	38
(2,3)	101	91	54

Result 2.

From these results we can conclude that:

- Increasing the holding cost to a higher value (from -1 to -10 in our case) decreases the order up to level drastically. This change was expected since our original model was trying to avoid loss sales at all costs because of its high cost. When we increased the holding cost, however; it stopped doing that and decreased its order-up-to levels.

- We can see that increasing the route costs in 3rd model also resulted in a decrease in the order-up-to levels. This result was also expected since the model will avoid choosing these routes and therefore reducing the order-up-to levels.

- Increasing the route cost for routes (1,2), (1,3), (2,3) made the model prefer routes (1), (2), (3) more instead. Basically, our model sometimes chose to bear the loss sales instead paying for the costly route options.