Artificial Intelligence Laboratory (CS-362)
End Semester Report
Group – AI-Beginners

Pushkaraj Kolekar (202051149) [1], Suyash Jain (202051189)[2], Prasoon Pathak (202051143)[3]
Dhruv Rashiya (202051153)[4]
Indian Institute of Information Technology Vadodara

CONTENTS

I. Problem 8

Problem Statement: Basics of data structure needed for state-space search tasks and use of random numbers required for MDP and RL.

A. Introduction of MENACE - by Donald Michie

MENACE learns to play noughts and crosses by playing the game repeatedly against another player. It improves its approach with each game until, after a predetermined number of games, it is nearly flawless and its opponent can do no better than draw or lose.It resembles a neural network in some ways because both start out randomly optimised. Reinforcement learning is a method of learning in which failure is "punished" and success or success-related "rewards" are "rewarded" for. Here, each matchbox symbolises a particular Noughts and Crosses board configuration..If each unique layout is considered then the number of boxes required would be very large. In order to lessen the boxes, Layouts that are mirror images of one another or are symmetrical are treated as a single box, keeping the necessary number of boxes to 304.

B. Training MENACE

All of the boxes initially contain coloured beads, each of which stands for a certain move or location on a board. The human player draws a bead at random from a box to symbolise the game's current state as comparable to the weights of a neural network, and MEN- ACE moves. Where MENACE moves depends on the hue of the bead that has been chosen from a certain box.

C. Cumulative Reward Equation

Beads in the box are adjusted when there is success or failure. At the end of each game –

- if MENACE loses, each bead MENACE used is removed from each box.
- If MENACE wins, three beads the same as the colour used during each individual turn are added to their respective box.
- If the game results in a draw, one bead is added to each box of the chosen colour.

Considering the above points the equation formed for the number of beads in the first box will be -

3 X Wins + Draws - Losses

After many games are played out, the matchboxes will have more of some beads than the others. This certainly tinkers with the probability of a bead being selected randomly which changes the probability of the next state after every move.

D. Analysis & Results

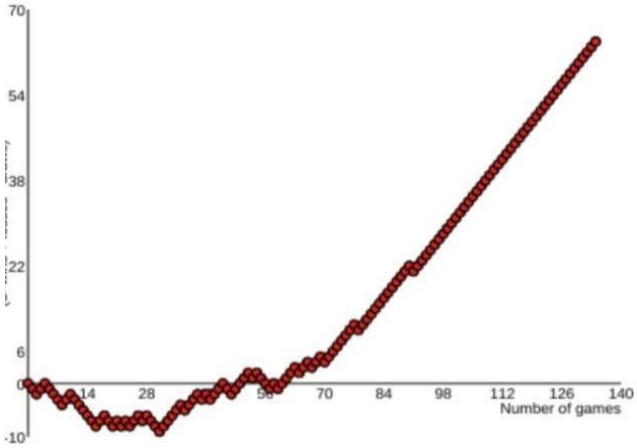


Figure: Variation of Reward Equation (Y axis) with number of games (X axis) for Menace against **Perfect Machine**

As we can see in the graph when MENACE plays a perfect-playing computer initially the value of the equation goes negative indicating the losses but later it increases gradually. Although the number of wins increase rarely but as a draw is considered positive to we observe the given result suggesting that MENACE has learned. The graph suggests that MENACE could never win against the perfect algorithm, but ended up drawing every time after about 90 games, making it equally as perfect as a computer.

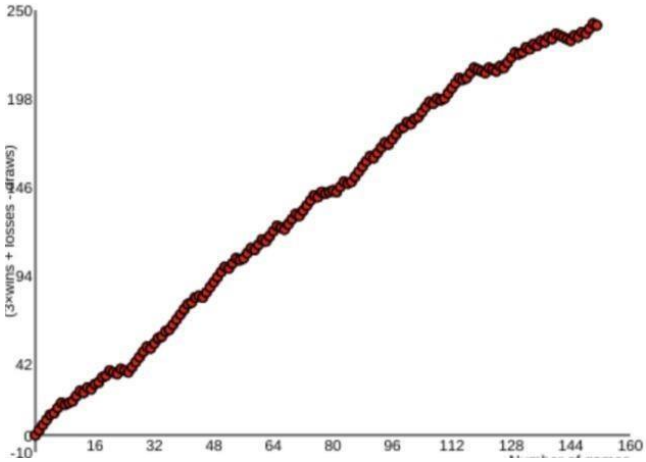
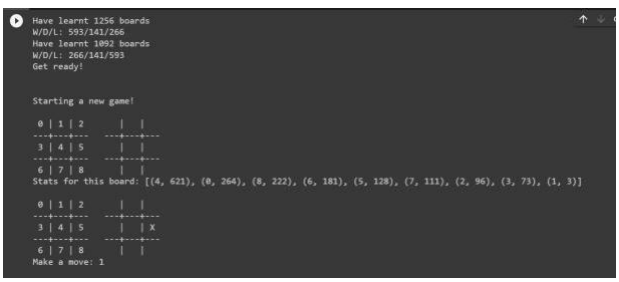
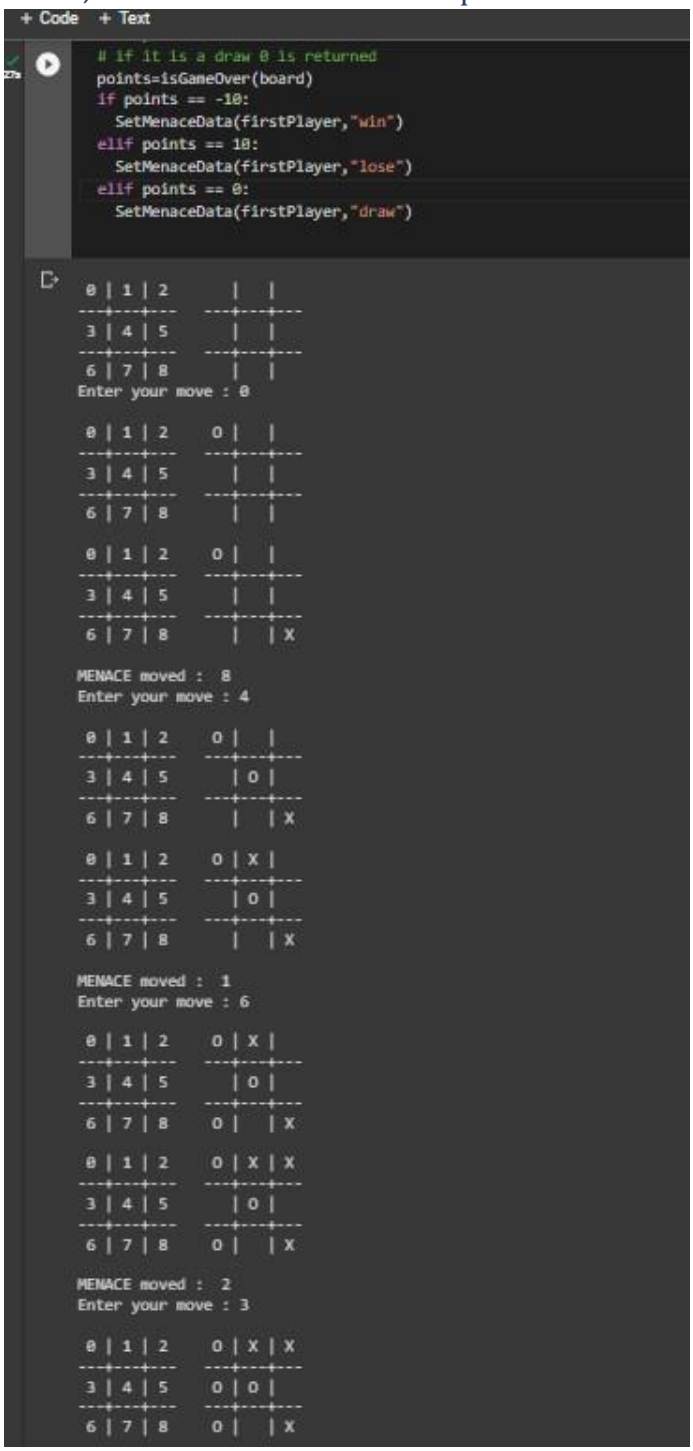


Figure: Variation of Reward Equation (Y axis) with number of games (X axis) for Menace against **Unintelligent Machine**

A. Algorithm

1) MENACE is made up of a series of match- boxes, each of which represents a different game board configuration. Each matchbox contains a variety of coloured beads, each of which represents a different move that MENACE could make.
2) When MENACE plays Tic-Tac-Toe, it selects a matchbox that represents the current game board configuration.
3) MENACE then draws a bead at random from the matchbox to represent its move for that turn.
4) After the game, MENACE is given a reward or a penalty based on the outcome. If MEN- ACE wins, it is rewarded (in the form of additional beads added to the matchbox). It is penalised if it loses or draws (in the form of beads removed from the matchbox).
5) Based on the feedback it receives from of the rewards and penalties, MENACE learns which moves are more likely to result in a win over time. It adjusts the number of beads in each matchbox accordingly, increasing the chances of selecting the best moves in subsequent games.
6) MENACE gradually improves its game playing ability and becomes increasingly dif- ficult to defeat through this trial-and-error process

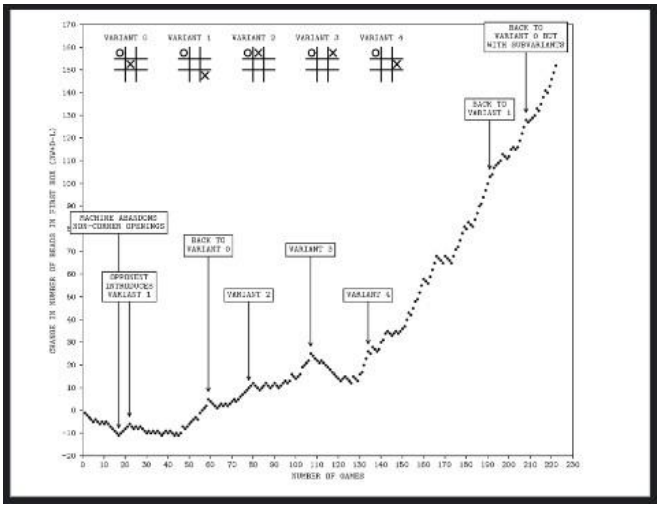




```
 0 | 1 | 2     X | O |
---+---+---   ---+---+---
 3 | 4 | 5       | O | X
---+---+---   ---+---+---
 6 | 7 | 8       |   |
Stats for this board: [(2, 7), (3, 3), (6, 3), (7, 3), (8, 3)]

 0 | 1 | 2     X | O |
---+---+---   ---+---+---
 3 | 4 | 5     X | O | X
---+---+---   ---+---+---
 6 | 7 | 8       |   |
Make a move: 7

 0 | 1 | 2     X | O |
---+---+---   ---+---+---
 3 | 4 | 5     X | O | X
---+---+---   ---+---+---
 6 | 7 | 8       | O |
You won!
Get ready!
```



On the other hand when MENACE plays a with a random picking opponent , the result is a near-perfect positive correlation which indicates that against an unintelligent player the tendency of loss for MENACE is very less although it makes random moves in the beginning still the equation never becomes negative.

II. Problem 9

Problem Statement: Understanding Exploitation- Exploration in simple n-arm bandit reinforcement learning task, epsilon-greedy algorithm. n-armed bandit.

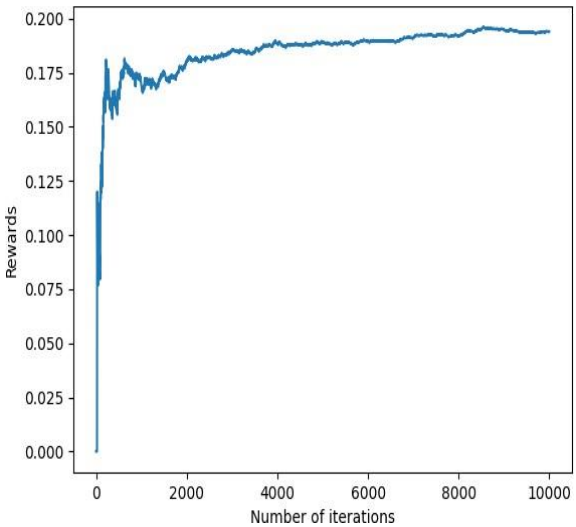
Algorithm 1 Epsilon Greedy Algorithm

Initialize, for $a = 1$ to k:
$Q(a) \leftarrow 0$
$N(a) \leftarrow 0$

Repeat forever:
$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$
$R \leftarrow bandit(A)$
$N(A) \leftarrow N(A) + 1$
$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$

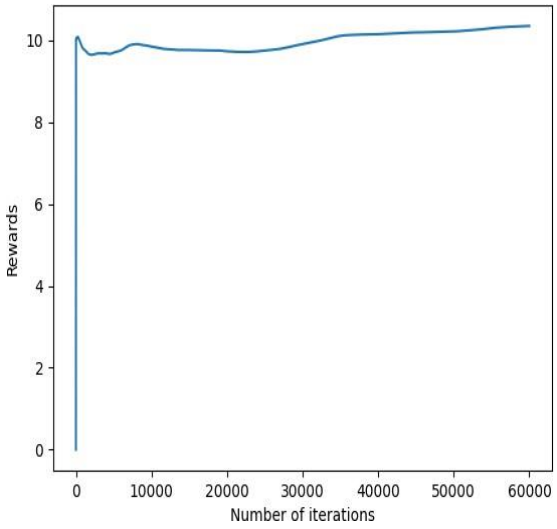
A. Binary bandit with stationary reward (epsilon greedy technique)

- This is a 2 arm-bandit where the rewards are 1 for success and 0 for failure.
- The exploration and exploitation are based on the Epsilon Greedy Algorithm.
- A random number between 0 and 1 is generated. If the generated number is greater than epsilon then exploitation is performed on the basis of the prior knowledge.
- In exploitation, the maximum Q value is selected and the actions which corresponds to maximum reward is performed. In the other case, exploration is done and the reward value corresponding to the action is given.



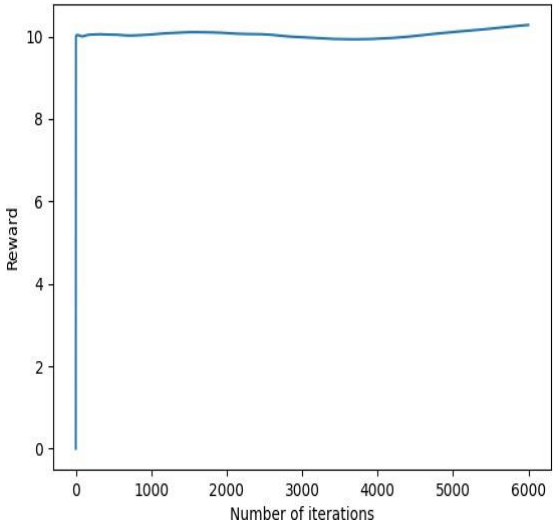
B. Develop a 10-armed bandit in which all ten mean- rewards start out equal and then take independent random walks

- This is a 10 arm-bandit where the mean-rewards is initialized with array of ones.
- The exploration and exploitation is based on the Epsilon Greedy Algorithm.
- A random number between 0 and 1 is generated. If the generated number is greater than epsilon, we perform exploitation on the basis of the prior knowledge otherwise exploration is performed.
- In each itertaion array whith length of 10 and whose val- ues has mean 0 and standard deviation 0.01 is generated.
- –The mean array is updated by adding the values of the generated array to the mean array.The updated mean array is used to give reward to every action.



C. n-arm (10) bandit with non-stationary reward (modified epsilon greedy technique)

- The only difference in this problem from Problem 2 is that in Problem 2 averaging method to update estimation of action reward was used whereas here more weight is given to current earned reward bu using alpha parameter.



From the above graphs we can conclude that the average reward in epsilon greedy is more than the simple algorithm.

I. Explanation

This problem is a well-known example of reinforcement learning and the exploration-exploitation tradeoff conundrum. The name is derived from the idea of a gambler faced with a row of slot machines (sometimes referred to as "one-armed bandits"), who must choose which machines to play, how many times to play each machine, in what order, and whether to keep playing the current machine or try a different machine. Each machine in the problem offers a random payout from a probability distribution unique to that machine, which is unknown at the outset.The gambler's goal is to pull enough levers to increase the total amount of winnings. At every trial, the gambler must choose between "exploration" to learn more about the expected payoffs of the other machines versus "exploitation" of the machine with the highest expected payoff.

A. Stationary problem

In our n-armed bandit problem, each of the n actions has an expected or mean reward given that that action is selected; let us call this the value of that action. We denote the action selected on time step t as A_t, and the corresponding reward as R_t. The value then of an arbitrary action a, denoted q (a), is the expected reward given that a is selected:

$$q_*(a) = E[R_t || A_t = a]$$

We assume that you do not know the action values with certainty, although you may have estimates. We denote the estimated value of action a at time step t as $Q_t(a)$. We would like $Q_t(a)$ to be close to q (a). In the multi-armed bandit problem, a thoroughly exploratory agent will sample all the bandits at a uniform rate and learn information about each bandit over time. The downside of such an agent is that it never uses this knowledge to enable itself to make better judgements in the future. On the other hand, a completely greedy agent will pick a bandit and stick with it for all eternity; it won't try other bandits in the system to see if they have better success rates to help it maximise its longterm rewards, so it is very limited in its thinking
at the moment, while the exploratory mechanism allows the agent to look for potential better bandits. .

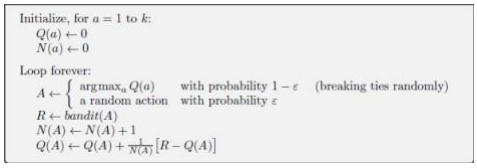


Fig. 1. Pseudocode for Stationary Complete Bandit Algorithm

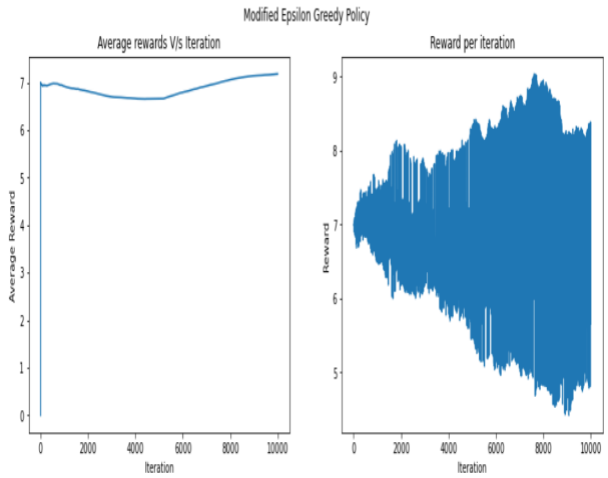
B. Non-stationary problem

C. In such cases it makes sense to give more weight to recent rewards than to long-past rewards. One of the most popular ways of doing this is to use a constant step-size parameter.

$Q_{n+1} = Q_n + \alpha[R_n - Q_n]$where $\alpha \in (0, 1]$.

I. OBSERVATIONS

Problem 1: We did two runs with different seed values to set different initial probabilities and then ran the Binary Bandit Algorithm we got the following results



II. RESULTS

The following conclusions can be drawn from the observations of problem 1.

- The algorithm has learned that action 2 is the best option for banditA and that action 1 is the best option for banditB.
- Our experiment highlights the importance to balance exploitation and exploration in reinforcement learning issues. Exploiting the available information is vital to maximise the benefit in the long term, even though it is critical to investigate novel activities in order to understand their reward distributions.
- As can be seen, the machine learned it non-deterministically via reinforcement learning, and the observed rewards are extremely close to the highest rewards that can be received.

The following conclusions can be drawn from the observations of problem 2.

- We looked at how the number of arms affected how well the algorithms performed. Our findings demonstrate that the algorithms' performance degrades as the number of arms rises because it becomes more challenging to sufficiently explore each arm.
- We were able to attain the highest average payout for the given random Gaussian distribution of the

probabilities and rewards in the N-arm Bandit prob- lem after a sufficient number of repetitions. This shows that the algorithms we utilised were successful in striking a good balance between exploitation and exploration.

The following conclusions can be drawn from the observa- tions of problem 3.

- We pick a variable called alpha to keep track of the weights assigned to the observed rewards when the rewards are not stationary. We prioritise recent incentives more highly. The graph also shows that the average reward received is quite close to the highest payout possible. Before evaluating the outcomes, we performed 10,000 iterations.

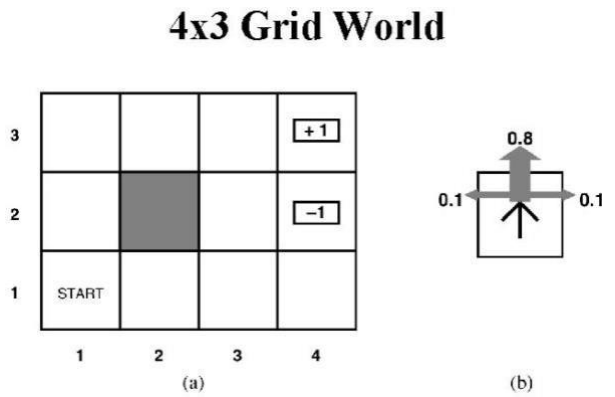
REFERENCES

Reinforcement Learning: An Introduction by R. Sutton and A. Barto (Second Edition) (Chapter 1)

III. PROBLEM 10

Problem Statement Understand the process of sequential decision making (stochastic environment) and the connection with reinforcement learning.

A. Suppose that an agent is situated in the 4x3 environment as shown in Figure. Beginning in the start state, it must choose an action at each time step. The interaction with the environment terminates when the agent reaches one of the goal states, marked +1 or -1.



Algorithm 2 Value Iteration: Learn function $J : X \rightarrow R$

Require:

Sates $X = \{1, \ldots, n_x\}$

Actions $A = \{1, \ldots, n_a\}$, $A : X \Rightarrow A$

Cost function $g : X \times A \rightarrow R$

Transition probabilities $f_{xy}(a) = P(y \mid x, a)$

Discounting factor $\alpha \in (0, 1)$, typically $\alpha = 0.9$

procedure VALUEITERATION(X, A, g, f, α)

Initialize $J, J' : X \rightarrow R_0^+$ arbitrarily **while** J is not converged **do** $J' \leftarrow J$ **for** $x \in X$ **do**

for $a \in A(x)$ **do**

$$Q(x, a) \leftarrow g(x, a) + \alpha \sum_{j=1}^{n_x} f_{xj}(a) \cdot J'(j)$$

for $x \in X$ **do**

$J(x) \leftarrow \min_a\{Q(x, a)\}$

return J

Value iteration computes the optimal state value function by iteratively improving the estimate of V(s). The algorithm initialize V(s) to arbitrary random values. It repeatedly updates the Q(s, a) and V(s) values until they converges. Value iteration is guaranteed to converge to the optimal values.

B. Find the value functions corresponding optimal policy for the following:

1) r(s) = -0.04

Value function Matrix			
-1.22	-0.82	-0.27	0.00
-1.45	0.00	-0.87	0.00
-1.53	-1.45	-1.22	-1.17

2) r(s) = -2

Value function Matrix			
-59.69	-46.00	-24.31	0.00
-65.39	0.00	-21.93	0.00
- 63.08	-52.78	-34.49	-20.74

Giving negative reward reflects Q(1,2) that the value function near the reward state is higher than away from the reward state, and it is expected behaviour of the value iteration policy, because that should only be the way to reach the end state *3) r(s) = 0.1*

Value function Matrix			
2.93	2.38	1.44	0.00
3.08	0.00	0.63	0.00
2.83	2.19	1.15	0.22

4) r(s) = 0.02

Value function Matrix			
0.54	0.54	0.45	0.00
0.47	0.00	-0.23	0.00
0.32	0.09	-0.21	-0.58

5) r(s) = 1

Value function Matrix			
29.78	23.13	12.48	0.00
32.44	0.00	10.30	0.00
31.10	25.76	16.42	9.21

while giving positive reward Q.(3,4,5) deflects the agent and force to stay in the state where it is, so in conclusion we can say that it is better to give reward in negative number, that helps agent to go towards the destination state and get maximum rewards, as we can see in (5) when reward = 1, the value at [1,1] is higher than [1,3], it means the agent is more stable at position [1,1] but in ideally it should be at position

[1,3]

C. Gbike bicycle rental with policy iteration and improvement

We have used Dynamic Programming to break up problem into smaller subsets and iteratively solve them. This can be done as we have a perfect model of the environment in the form of an Markov Decision Process. The goal here is to setup the environment of the MDP defining the states, actions, probabilities, and rewards and further use Policy Iteration to solve the problems.

Policies are a mapping of the actions that an agent takes from a particular state. Many different actions are possible for the agent, so to solve this problem, it needs to know which action will result in the highest long-term return.

1) Policy Iteration

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \text{argmax}_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Algorithm of Policy Iteration

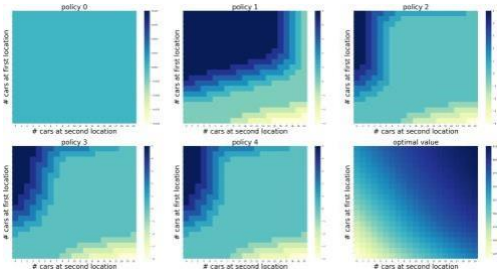
Policy Evaluation is the process of taking a policy, and calculating the return of every single state based on following a particular policy. Once we arrive at a true value function for our policy, we can attempt to improve it.

Policy Improvement is the act of looking at all actions we could take from a given state, acting greedily by choosing the action that gives the highest return with respect to our value function.

Every time we arrive at the true value function for a policy, we determine how much value we should expect to return at each state given a particular policy. This helps us in making a better policy. At the beginning, we randomized our policy but with a true value function, we can exploit the values by changing our policy to choose the best action that will return the highest value from a given position.

Improving the policy involves testing actions at each state and choosing the best action among them. Unlike evaluation, here we iterate through all the actions and then have a list of returns to look over.

The system go through these two processes again and again. It finds the true value function for a policy, and then improve the policy based on the value function, before going back and refining the value function again, whereby we improve the policy and keep doing this until it arrives at the optimal policy for our solution.



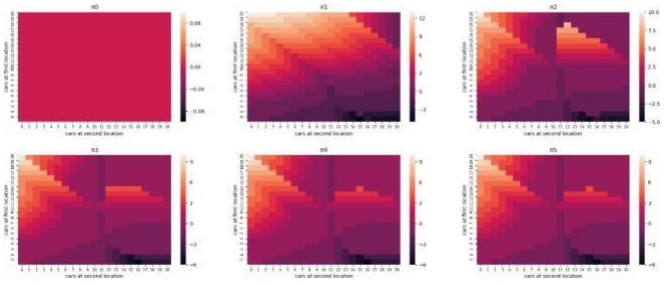
Improvement of policy

As we can see the graph, the left figure of given graph is shown for the initial state of the car which is 0 and 0 both the sides, and then iterating over new policy evaluation and continuous policy improvement, we end up being in the last 6th state, where we can see the optima has been occurred.

D. Synchronized Gbike bicycle rental with policy iteration and improvement

Adding non-linearities to the original rental problem • One of Jack's employees at the first location rides a bus home each night and lives near the second location. She is happy to shuttle one gbike to the second location for free. Each additional gbike still costs 2, as do all cars moved in the other direction.

- In addition, Jack has limited parking space at each location. If more than 10 gbike are kept overnight at a location (after any moving of gbike), then an additional cost of 4 must be incurred to use a second parking lot (independent of how many gbike are kept there).



Improvement of policy

The above graph also shows the similar property as the above one, we are trying to optimize the policy and after 5 iteration we are getting the optimal policy which can not be improved.

E. Conclusion

We solved the above problems using Dynamic Programming. We stored the intermediate value and policy matrices and used them in our policy evaluation and improvement functions. In order to use Bellman updates though, we need to know the dynamics of the environment, just like we knew the probabilities for rewards and the next states in the rental example. If we can only sample from the underlying distribution and don't know the distribution itself, then Monte Carlo methods can be used to solve the corresponding learning problem.

1) Value Function :

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

s is current state , a is the action and s' is the next state, r is the reward.

B. Gbike Rental Problem and Algorithm

The MDP formulation of the Gbike Rental Problem is as follows -

1) State: The states can be represented as -

$$[b^1_t, b^2_t]$$

$$b^i_t \in \{0, 1, ..., 20\}$$

b^1_t and b^2_t represent the number of bikes available at location 1 and location 2 respectively. *2) Action:* Sign convention -

If bikes transferred from location 1 to location 2 → − +ve

If bikes transferred from location 2 to location 1 − → -ve

The action $A(s_t)$ is in the range -

$$\{-min(min(5, b^2_t), 20 - b^1_t), min(min(5, b^1_t), 20 - b^2_t)\}$$

3) Transition Probability and Reward: By performing action $A_t = A(s_t)$, the state is transfered from

$$[b^1_t, b^2_t] \text{ to } [b^1_{t+1}, b^2_{t+1}]$$

The chance node being

s_t, A_t with reward $-2|A_t|$.

After performing the action A_t, the number of bikes remaining at the locations are as follows -

$$b^1_t - A_t, b^2_t + A_t$$

The next day, bikes are rented and returned, changing the number of bikes at each location, the final state is as follows -

$$[b^1_{t+1} = min(b^1_t - A_t - (min(b^1_t - A_t, r^1_{t+1})) + R^1_{t+1}, 20), b^2_{t+1} = min(b^2_t - A_t - (min(b^2_t - A_t, r^2_{t+1})) + R^2_{t+1}, 20)]$$

Where, $requests \rightarrow [r^1_{t+1}, r^2_{t+1}]$

$returns \rightarrow [R^1_{t+1}, R^2_{t+1}]$

Transition Probability -

$$P(r^1)P(r^2)P(R^1)P(R^2)$$

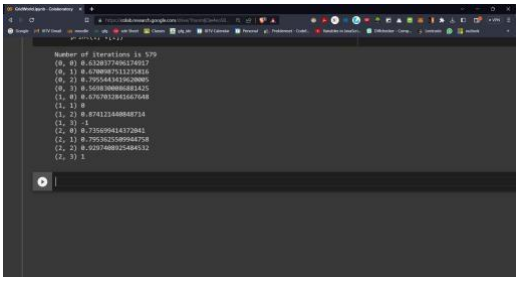
Where, Probability distribution is Poisson Distribution

Reward - from $s_t \rightarrow s_{t+1}$

$$10min(b^1_t - A_t, r^1_t) + 10min(b^2_t - A_t, r^2_t) - 2|A_t|$$

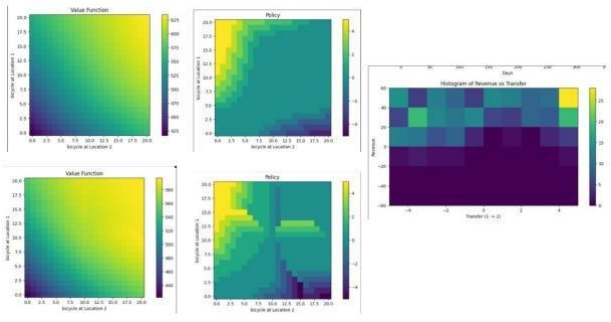
Initialization: Set an initial policy, which specifies the action to be taken at each state of the MDP. Policy evaluation: Given the current policy, compute the value function, which represents the expected total reward that can be obtained by following the policy from each state. Policy improvement: Modify the policy to be greedy with respect to the current value function, i.e., choose the action that maximizes the expected total reward from each state. Repeat steps 2 and 3 until convergence.

I. OBSERVATIONS



A. Problem 2 and Problem 3

The MDP is formulated, and algorithm is mentioned in the theory section.



II. RESULTS

From the observations of problem 1, we can assess that

- By iteratively using the Bellman equation and updat- ing the value function until convergence, the optimal value function and optimal policy can be calculated.
- The formulation of Value function has been done.
- An illustration of how MDPs can be used to model decision-making difficulties and how value iteration can be utilised to determine the best course of action is provided by the Gridworld problem. The formulation of MDP for problem 2 has been done.

From the observations of problem 3, we can assess that

- We pick a variable called alpha to keep track of the weights assigned to the observed rewards when the rewards are not stationary. We prioritise recent incentives more highly. The graph also shows that the average reward received is quite close to the highest payout possible. Before evaluating the outcomes

IV. ACKNOWLEDGMENT

We are highly indebted to Dr Pratik Shah for his guidance and constant supervision as well as for providing necessary information for completion of assignment problems. We would like to express our special gratitude and thanks to Teaching Assistants who helped us in resolving our doubts and putting forward helping hands.

V. GITHUB

LINK Group – AI_Beginners

REFERENCES

[1] Experiments on the mechanization of game-learning
[2] Menace: the Machine Educable Noughts And Crosses Engine
[3] How 300 Matchboxes Learned to Play Tic-Tac-Toe Using MENACE
[4] K Armed Bandit Reinforcement Learning by Richard Sutton and Andrew Barto Chapter 4