Indian Institute of Technology Delhi

COL333: Intro. to Artificial Intelligence
Assignment 3: Planning in the Taxi Domain

Devanshi Khatsuriya: 2019CS10344

Ishita Chaudhary: 2019CS10360

Professor Rohan Paul

Department of Computer Science & Engineering

Contents

1 Part A: Computing Polices

1.1 Formulating the Taxi Domain Problem as MDP

(a) The Taxi Domain can be formulated as an MDP with attributes as follows:

- State Space: State is represented as a 3-tuple (location of taxi, location of passenger, whether passenger is in taxi [boolean]). The location of taxi and the location of passenger are both represented as a 2-tuple (row index of location in grid, column index of location in grid). There are 25 locations in the grid, so the number of states in which passenger is not in taxi (states with boolean = False) are 25×25, and the number of states in which passenger is in taxi are 25. So, the total number of states is $25 \times 26 = 650$.

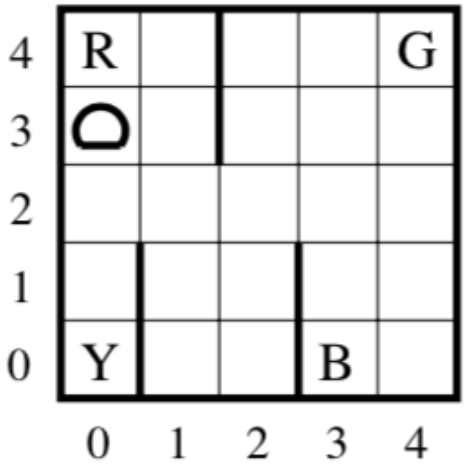

Figure 1: Taxi Domain Grid

- Initial State: In the initial state, the location of taxi is a randomly chosen grid location, the location of passenger is a randomly chosen depot (depots are locations denoted by R, B, G and Y in Figure 1) and the passenger is assumed to not be in the taxi (boolean = False).
- Goal State: A destination location is chosen among all depots (other than initial depot of passenger). The goal state then is (destination depot, destination depot, False), that is, both taxi and passenger at the destination depot with passenger not in the taxi.
- Action Space: In each state (except the goal state), 6 actions are available. These are 4 navigation actions - N, E, S and W - in which the taxi moves in the corresponding direction, and 2 actions corresponding to PICKUP and PUTDOWN of passenger by taxi.
- Transition Model: Navigation action on a state leads to next state with the same boolean and the location is in expected direction with probability 0.85 and other directions with probability 0.05. If there is a wall in any direction, then probability of next state having the same taxi location as current state increases by the probability of going in the walled

direction. PICKUP action for states with same taxi and passenger location leads to next state having same location and boolean True, while PUTDOWN for such states leads to boolean False, both with probability 1. PICKUP and PUTDOWN for other states leads to next states being the same as current state with probability 1.

- Reward Model: All navigation actions give a reward of -1. PICKUP and PUTDOWN give a reward of -10 for states where location of taxi $\neq$ location of passenger. PUTDOWN at (destination location, destination location, True) gives a reward of +20. PICKUP and PUTDOWN at all other states give a reward of -1.

(b) Implemented a simulator of the Taxi Domain MDP (Refer to *class TaxiDomain* in code).

1.2 Implementing Value Iteration

(a) Implemented Value Iteration to find the optimal policy for an initial state, goal state initialization of the Taxi Domain MDP. The max-norm distance between successive value functions is used to determine convergence (converged if max-norm distance $\leq$ epsilon$\times$(1$-$discount)/discount). With epsilon = 0.01 and discount = 0.9, value iteration converges in 35 iterations.

(b) Value iteration is run for discount factor values in $[0.01, 0.1, 0.5, 0.8, 0.99]$ (with epsilon fixed at 0.01). The plot of max-norm distance between successive value functions as iterations progress for different discount factors is shown in Figure 2.
Observations. From the plot, we see that convergence is faster for lesser discount factors.

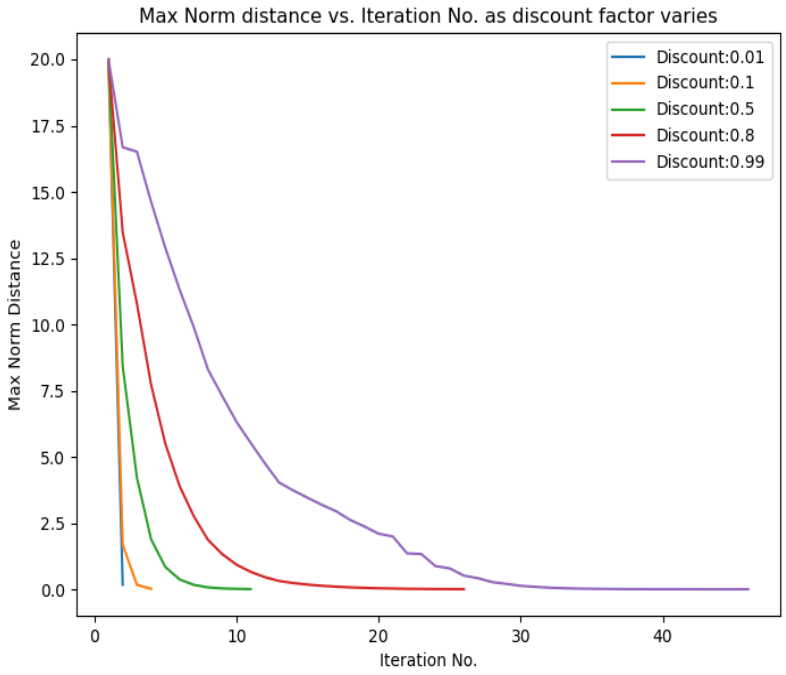

Figure 2: Part A: 2(b) Value Iteration

This is because as discount factor decreases, the dependence of the expected utility (or value) of a state on future rewards and transitions decreases, which depends on states that

are farther away. So, the lesser the discount factor, the more the value of each state depends only on nearby states, which in turn will take lesser iterations to learn (converge). This is because in the k^{th} iteration, the states that are upto k steps away have been considered in the value estimate of a state.

(c) The Value Iteration algorithm is tested on the Taxi Domain Problem for two different discount factors (gamma), 0.1 and 0.99. We analyse the first 20 state-action sequences for both these values, with error tolerance epsilon set to 0.01. These are shown in figures 3 and 4.
The initial locations were as follows:

- Passenger Location: Depot B (0, 3)
- Taxi Location: (1, 1)
- Destination Location: Depot Y (0,0)

The arrow denotes the flow of states, labelled with the action taken and followed by the resulting state. T denotes an empty taxi (with passenger at its initial location), and P denotes the passenger is inside the taxi.
Figures 3 and 4 show the state-action sequences for discount factor = 0.1 followed by discount factor = 0.99.

Observations:

- The agent does not even pick up up the passenger in the first 20 steps with gamma = 0.1, however with gamma = 0.99, the agent picks up the passenger and drops them off at the destination in just 17 iterations.
- The algorithm with a considerably smaller discount factor tends to maximise its immediate reward, whereas the one with a higher discount factor maximises its total reward.
- As a result, the one with gamma = 0.1, picks actions which do not necessarily reflect the optimal choice, and hence are random in nature. But the actions followed by the policy with gamma = 0.99 are quite optimal, due to the fact it tends to achieve the +20 reward on PUTDOWN at destination location.
- Even though the policy followed by the gamma = 0.1 is not optimal, it never chooses the PICKUP or PUTDOWN action when the passenger is not in the same cell as taxi. Because this will give a reward -10, and it is trying to maximise its immediate reward.

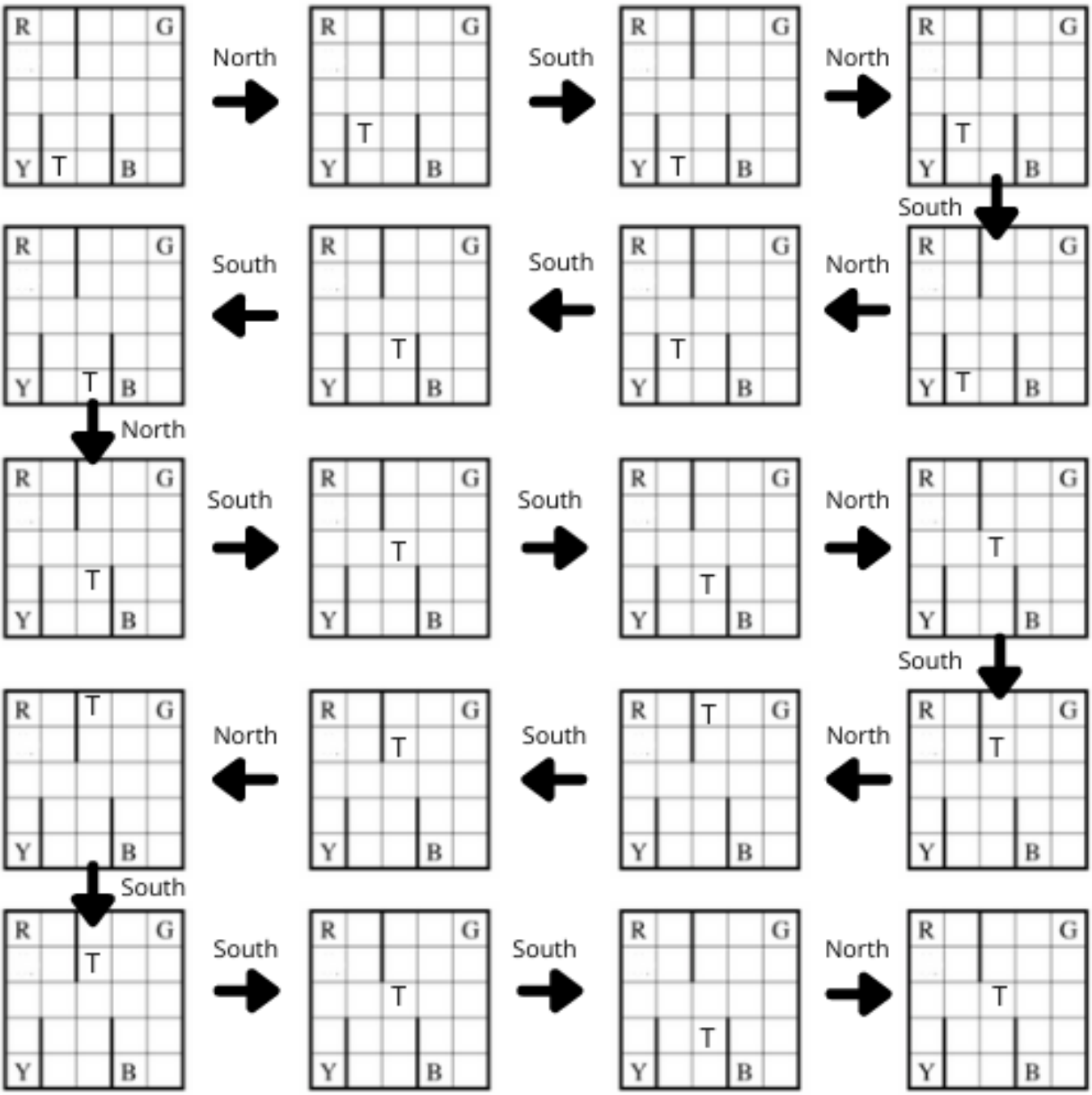

Figure 3: Part A: 2(c) gamma = 0.1

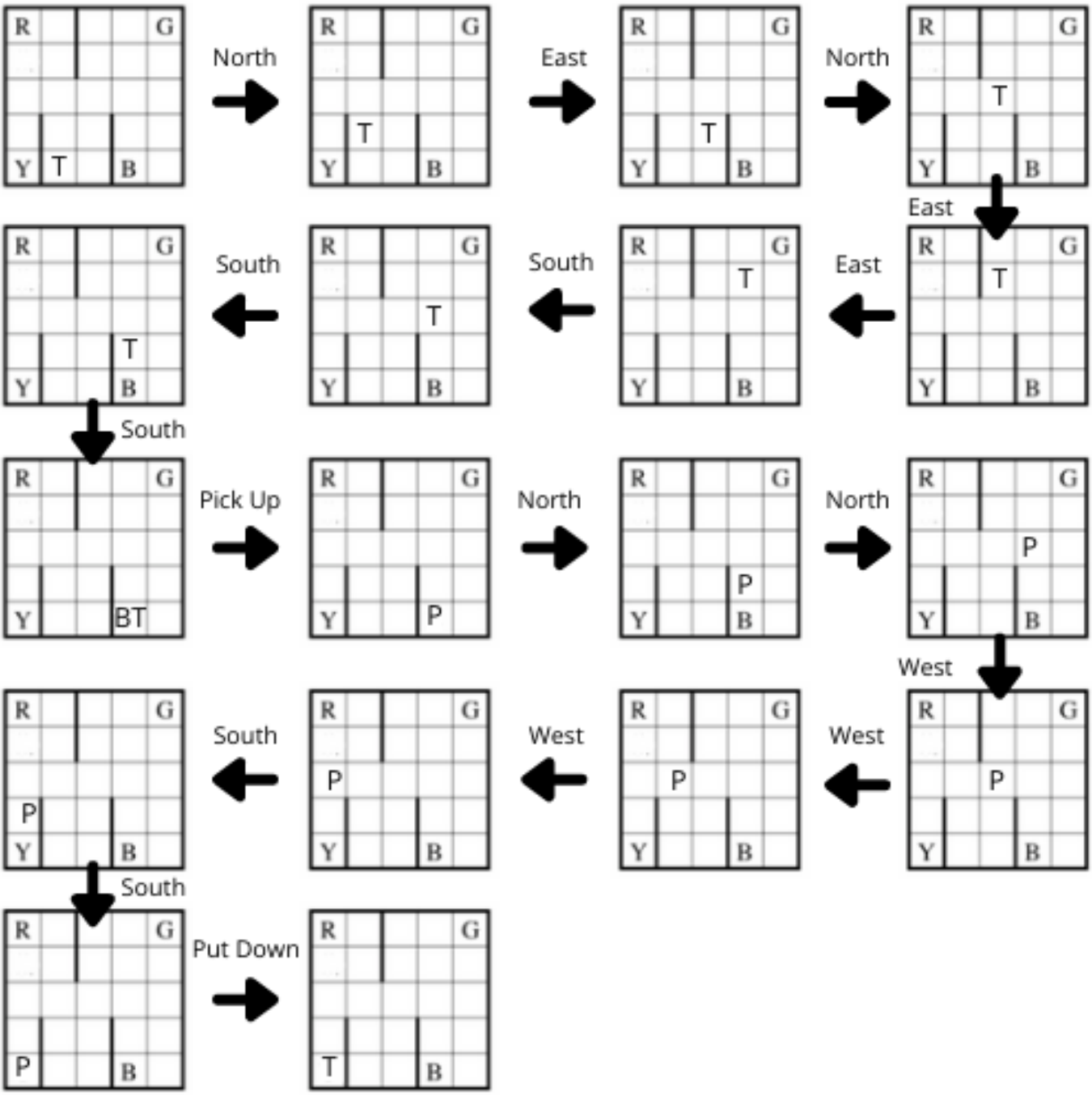

Figure 4: Part A: 2(c) gamma = 0.99

1.3 Implementing Policy Iteration

(a) Implemented two methods for policy evaluation:

- Linear Algebra Method: The values of states corresponding to a policy π can be represented as a system of n linear equations in n variables (variables are $V^\pi(s)$ for $s \in states$ that denotes the value of state s under policy π), where n is the number of states. For each state s, we have one equation corresponding to (1). The values of states can be obtained by solving this system of equations. We do this using *np.linalg.solve*.

$$V^\pi(s) - \sum_{s'} T(s, \pi(s), s') \times \gamma \times V^\pi(s') = \sum_{s'} T(s, \pi(s), s') \times R(s, \pi(s), s') \quad (1)$$

- Iterative Method: In this method, we iteratively update estimates of values of states till convergence (estimated values are assumed to be converged when the max-norm distance between successive estimates becomes less than $1e^{-4}$). The values are initialized as in (2) and updated in the $k+1^{th}$ iteration as in (3), where $V_k^\pi(s)$ denotes the estimate of value of state s under policy π in the k^{th} iteration.

$$V_0^\pi(s) = 0 \quad (2)$$

$$V_{k+1}^\pi(s) = \sum_{s'} T(s, \pi(s), s') \times [R(s, \pi(s), s') + \gamma \times V_k^\pi(s')] \quad (3)$$

The Linear Algebra method takes $O(n^3)$ time, while the Iterative method takes $O(kn^2)$ time, where n is the number of states and k is the number of iterations. The space complexity of iterative method is only $O(n)$, while the space complexity of linear algebra method is $O(n^2)$. When the number of states is very large, the linear algebra method might perform worse. If the discount factor is more, iterative policy evaluation takes longer to converge, so linear algebra method might be better. For 3(b) below, we observe that policy iteration with the linear algebra method takes 4.77 seconds while the iterative policy evaluation takes 9.25 s.

(b) Implemented Policy Iteration to find the optimal policy for an initial state, goal state initialization of the Taxi Domain MDP. Policy iteration converges if policy remains unchanged after an iteration. In addition to this, we implement another sufficient condition for convergence, which is if the max-norm distance between evaluated values of consecutive policies is less than epsilon ($= 1e^{-4}$).

Policy iteration is run for discount factor values in $[0.01, 0.1, 0.5, 0.8, 0.99]$ (with epsilon fixed at $1e^{-4}$). Policy loss is defined as the max-norm distance between evaluated values for the current policy and the evaluated values for the optimal policy. The plot of policy loss as iterations progress for different discount factors is shown in Figure 5. The left plot uses iterative policy evaluation while the right plot uses the linear algebra method to evaluate policy.

Observations. We see that convergence is faster for lesser discount, because of the same reason as 1.2(b) that for lower discount factors, value of a state depends more only on nearby states, which inturn is faster to learn. However, lesser discount leads to policies in which the taxi does not learn to PICKUP the passenger and PUTDOWN at the destination because the future rewards are disregarded heavily, as we saw in 2(b). Additionally, we see that policy iteration takes lesser iterations to converge as compared to value iteration.

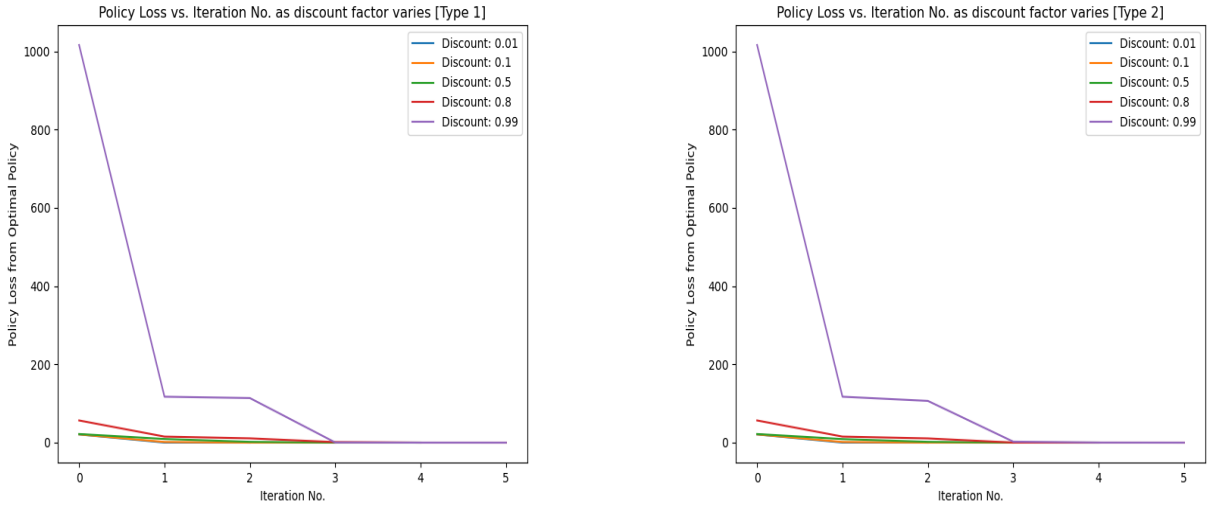

Figure 5: Part A: 3(b) Policy Iteration (using iterative and linear algebra policy evaluation)

2 Part B: Incorporating Learning

2.1 Implementation of Different Algorithms

Implemented the following model free reinforcement learning approaches to learn the optimal policy for a given goal state initialization of the Taxi Domain MDP:

(a) Q-learning with the following 1-step greedy look-ahead update:

$$Q(s,a) \longleftarrow (1-\alpha)Q(s,a) + \alpha[r + \gamma\ max_{a'}Q(s',a')] \tag{4}$$

Here $Q(s,a)$ is the current estimate of the Q-value of (s,a) pair, r and s' denote the reward and the next state on executing action a at state s, α is the learning rate, γ is the discount factor. All $Q(s,a)$ values are initialized to 0.
Also implemented an epsilon-greedy approach with fixed epsilon for exploration. Under this, the action to be taken in any state while training is chosen to be the a random action with probability epsilon, and the one with the maximum $Q(s,a)$ value with probability 1−epsilon.

(b) Q-learning with decaying exploration rate: The same Q-learning as above but the epsilon-greedy approach has decaying a exploration rate that varies as:

$$\text{Exploration Rate} = \frac{\text{epsilon}}{\text{No. of updates done so far}} \tag{5}$$

where 'No. of updates done so far' represents the no. of Q-learning updates of the form (4) done till the present training sample.

(c) State-Action-State-Action-Reward (SARSA): SARSA does Q-value updates as follows:

$$Q(s,a) \longleftarrow (1-\alpha)Q(s,a) + \alpha[r + \gamma\ Q(s',a')] \tag{6}$$

Here a' is the action that is actually taken at state s'.

(d) SARSA with decaying exploration rate: The same SARSA-learning as above but the epsilon-greedy approach has decaying a exploration rate that varies as (5).

All these can be trained for a fixed number of episodes. An episode involves initializing the initial state (location of taxi, location of passenger, False) and the agent taking action till the goal state is reached or a limit of 500 actions is reached, while updating the Q value estimates on taking each action using the update equations in (4) or (6).
Note that the goal state (destination location, destination location, False) is fixed among all episodes, so that on training, the Q-values leads to a policy that moves towards the fixed goal state.

2.2 Evaluation of All Four Algorithms

To evaluate all the four approaches on common grounds, the algorithms explained above are trained for 2000 episodes, with initial learning rate as 0.25, discount factor as 0.99, and exploration rate as 0.1. Further, the average discounted reward for each is calculated by taking average of accumulated reward over 20 episodes. the evaluation episodes also contain the learning step (q-values are updated in this episode too).
The graph obtained for average discounted reward against the episode number is given in Figure 6. The average discounted reward obtained at the end of training for each algorithm is reported below:

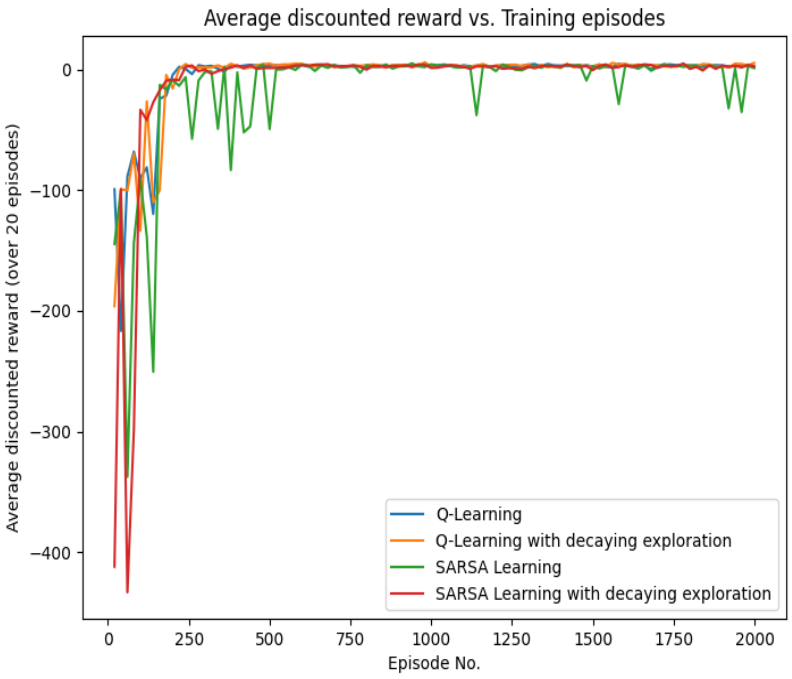

Figure 6: Part B, 2: Analysis of All Four Approaches

- Q-Learning: 2.2403
- Q-Learning with decaying exploration: 5.2782

- SARSA: 1.0389
- SARSA with decaying exploration: 2.4782

Observations and Inferences:
The decaying exploration approaches converge to somewhat higher values (as reported above) than the algorithms which explore every state. Because it is preferable to exploit instead of explore after we have a fairly good estimate of Q-values. Excessive exploration lowers the utility of an approach because unfavourable states are visited instead of the states that are most likely to maximise the reward.
Also, it can be noted that the Q-learning and Q-learning with decay both converge much rapidly than the SARSA Learning, which takes around 600 episodes to converge, whereas the Q-Learning algorithms converge around 250 episodes. This is due to the fact that in Q-learning, we select the best Q-value of the next state in order to update the Q-value in the first state. This is more efficient than merely utilising the Q-value associated with the next action taken, which is the approach of SARSA algorithm.

2.3 Analysing the Best Algorithm

As we observed in the previous section, the Q-Learning algorithm with decaying exploration rate proves to be the best approach for this problem. The taxi agent was trained for 2000 episodes for the destination location at T (0, 0).
Further, the policy was executed for five different instances, initial locations and final rewards reported as below:
1. Passenger Location: (0, 3); Taxi Location: (3, 1); Accumulated Discounted Reward: -2.29
2. Passenger Location: (0, 3); Taxi Location: (2, 1); Accumulated Discounted Reward: 3.81
3. Passenger Location: (4, 4); Taxi Location: (4, 1); Accumulated Discounted Reward: -4.70
4. Passenger Location: (0, 3); Taxi Location: (3, 4); Accumulated Discounted Reward: 2.28
5. Passenger Location: (0, 3); Taxi Location: (1, 0); Accumulated Discounted Reward: 1.52

Observations: Over the course of these five instances, the average accumulated discounted reward is 0.62. The policy we learned after training allows the agent to pick up and drop off passengers at their destinations over every run, and the average accumulated discounted reward is positive. Hence, we can conclude that the agent appears to be capable of performing successfully in any configuration of initial positions of taxi and passenger.

2.4 Varying the Learning Rate and Exploration Rate

In this part, we vary the learning rate and exploration rate for Q-Learning algorithm (without decaying exploration rate).
First, keep the learning rate (alpha) constant and equal to 0.1 and vary the exploration rate (epsilon) as in 0, 0.05, 0.1, 0.5, 0.9. The plot of average (over 20 episodes) accumulated reward against episode number (2000 episodes) obtained is given in Figure 7.

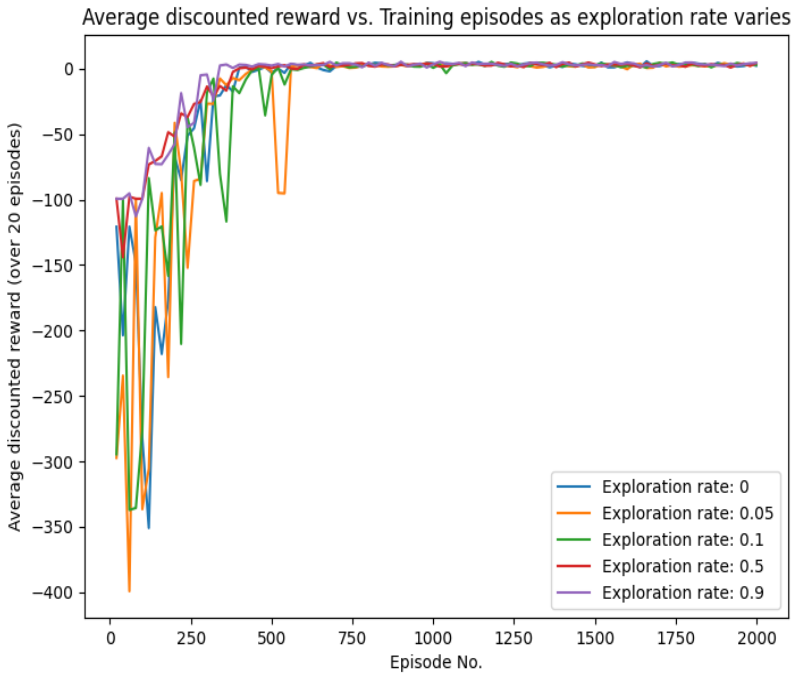

Figure 7: Part B, 4: Varying exploration rate

Observations and Inferences:

- The agent's performance in the early phase, i.e., with fewer training episodes, increases when the exploration rate is increased from 0 to 0.5. Because the agent's pace of learning will be slow at first if it does not explore states.
- However, when the exploration rate is further increased to 0.9, the agent's performance degrades relatively. This is because the agent is now selecting random actions far too frequently and failing to utilize the actions with the highest predicted Q value, resulting in a reduced accumulated reward.
- But as the training episodes are increased, the value of exploration rate doesn't affect the convergence. This is because there is no need for research after the Q values are learned, and hence is almost identical.
- Of all the given choices, we find that an exploration rate of 0.5 is most appropriate for solving this MDP.

Now, keeping the exploration rate (epsilon) constant and equal to 0.1 and we vary the learning rate (alpha) as in 0.1, 0.2, 0.3, 0.4, 0.5. The plot of average (over 20 episodes) accumulated reward against episode number (2000 episodes) obtained is given in Figure 8.

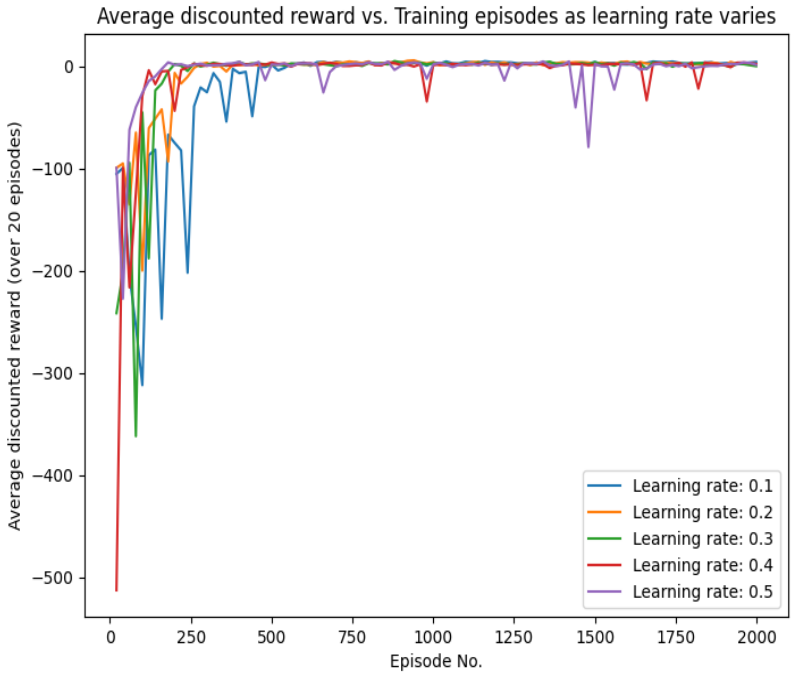

Figure 8: Part B, 4: Varying learning rate

Observations and Inferences: The pace at which convergence is attained rises when the learning rate increases from 0.1 to 0.5. It takes more than 500 training episodes for the agent with 0.1 learning rate to attain a non-negative average utility, whereas it takes roughly 250 for the one with 0.5 learning rate. This is because the Q values are updated faster during the learning phase for greater values of learning rates.

2.5 Extension of Taxi Domain Problem

Figure 9 indicates the 10x10 Taxi Domain Problem. We used Q-Learning with decaying exploration rate to solve this, with an exploration rate of 0.2, discount factor of 0.99, and a learning rate of 0.5. The above parameters were chosen after observing their results in the previous parts, and varying them to get better outcome.

The graph in Figure 10 depicts the average accumulated discounted reward obtained against training episode number. We can observe that the agent underperforms in the first 2500 episodes or so, but after around 6000 episodes, the reward begins to converge and becomes more steady as the agent is further trained.

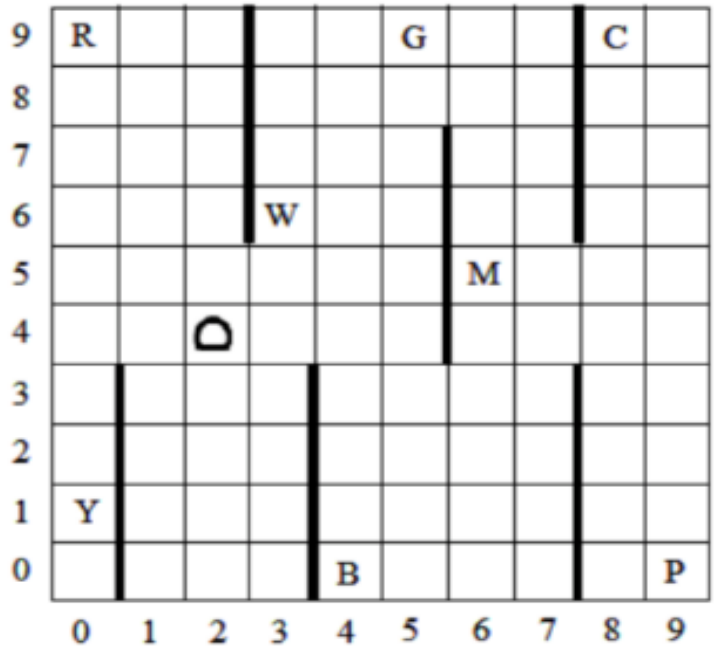

Figure 9: Part B, 5: 10X10 Taxi Domain Grid

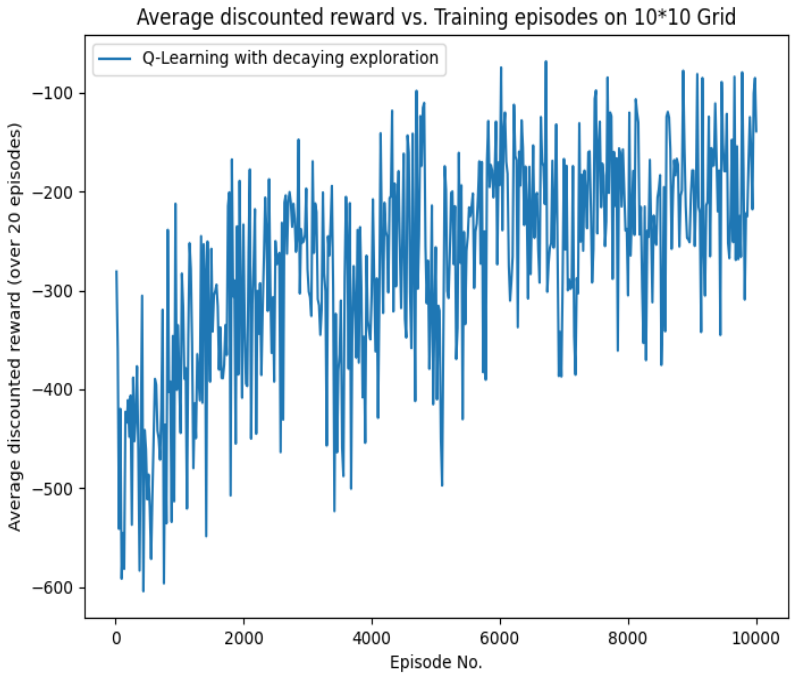

Figure 10: Part B, 5: Average Discounted Award vs. Training Episodes of Extended Taxi Domain Problem

The list below presents the results obtained for different instances when the learner is trained for 10, 000 episodes on a TDP where destination location is B (0, 4).

1. Passenger Location: (6,3); Taxi Location: (2,1); Accumulated Reward = -99.34
2. Passenger Location: (0,9); Taxi Location: (2,7); Accumulated Reward = -99.34
3. Passenger Location: (6,3); Taxi Location: (4,7); Accumulated Reward = -99.34
4. Passenger Location: (1,0); Taxi Location: (6,9); Accumulated Reward = -24.42
5. Passenger Location: (1,0); Taxi Location: (6,8); Accumulated Reward = -16.43
6. Passenger Location: (9,5); Taxi Location: (9,5); Accumulated Reward = 3.20
7. Passenger Location: (5,6); Taxi Location: (4,4); Accumulated Reward = -99.34

The average accumulated reward over 7 instances is -62.14.