# CS6700: Reinforcement Learning

# Programming Assignment 2

Srikar Babu Gadipudi, EE21B138
Shreya .S. Ramanujam, EE21B126

April 20, 2024

# Contents

# 1 Environment Description

The environment used in this assignment is Gymnasium Gym's Taxi-v3 environment, shown in Figure 1.

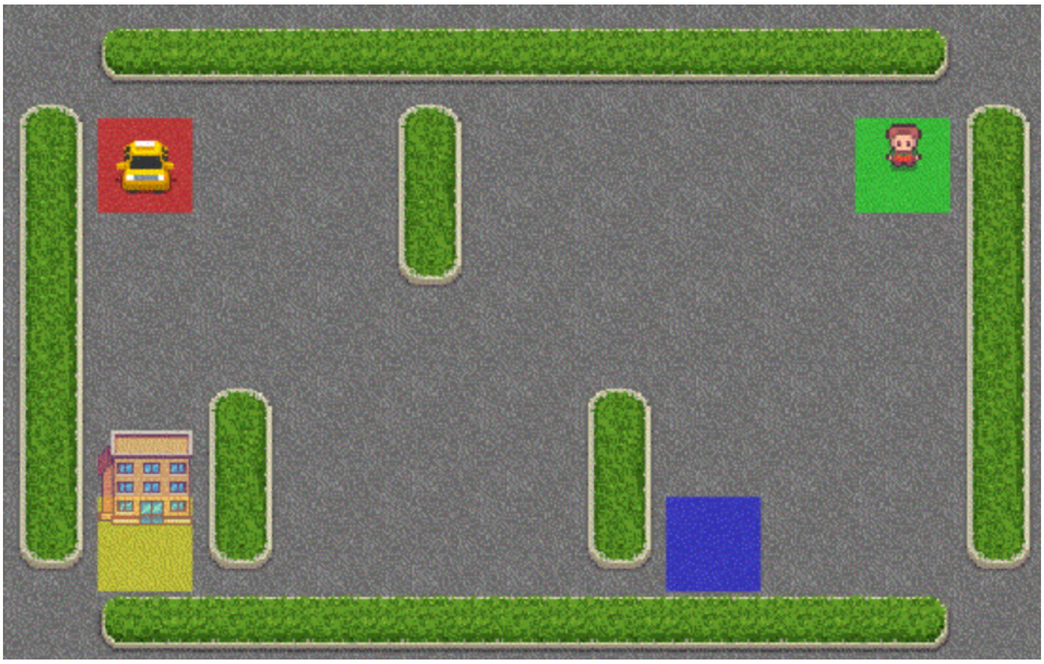


Figure 1: Taxi Environment

There are four designated locations in the grid world indicated by R(ed), G(reen), Y(ellow), and B(lue). When the episode starts, the taxi starts off at a random square and the passenger is at a random location. The taxi drives to the passenger's location, picks up the passenger, drives to the passenger's destination (another one of the four specified locations), and then drops off the passenger. Once the passenger is dropped off, the episode ends.

There are 500 discrete states since there are 25 taxi positions, 5 possible locations of the passenger (including the case when the passenger is in the taxi), and 4 destination locations. An observation is an integer that encodes the corresponding state. The state tuple can then be decoded with the "decode" method. Each state is represented by a tuple (x, y, p, d) where (x, y) is the location of the taxi on the grid, p denotes the location of the passenger (0: R, 1: G, 2: Y, 3: B, 4: in taxi) and d denotes the drop location (0: R, 1: G, 2: Y, 3: B).

There are 6 deterministic actions, 4 actions to navigate the taxi and 2 actions to pick and drop a passenger at any location in the 5x5 grid.

The rewards are such that the agent receives -1 per step unless another reward is triggered, +20 for delivering the passenger at the right location and -10 for executing 'pick' and 'put' actions illegally.

# 2 Option Description

Options are pre-defined behaviors that the agent can choose alongside standard actions like 'pick' and 'put'. In this case, options 'R', 'G', 'Y', and 'B' represent taking the taxi to their respective destinations. Each option has its own policy, learned through a simple Q-learning algorithm. This policy dictates how the agent navigates once a specific option (e.g., ‘R’) is chosen.

The agent can choose from six actions at each step: ‘pick’, ‘put’, or any of the four options (‘R’, ‘G’, ‘Y’, or ‘B’). To determine the best action we employ and compare algorithms: SMDP Q-learning and Intra Option Q-learning. These algorithms analyze the current state (taxi location, passenger status) and predict the best action (including choosing an option) to maximize future rewards (reaching destinations efficiently).

An important aspect of options is their termination condition. Each option ends once the taxi reaches its designated location. For example, option ‘R’ ends when the taxi arrives at the top-left corner of the grid (x=0, y=0).

## 3 SMDP Q Learning

The first algorithm we use in this assignment is SMDP Q Learning. The pseudocode is provided in 1.

### 3.1 Algorithm

**Algorithm 1:** SMDP Q Learning

```
Initialize $\alpha$ (learning rate) and $\gamma$ (discount factor)
Initialize Q table
for every episode do
    while episode not done do
        Select action ($a_t$) from state ($s_t$) using a policy derived from Q (e.g.
          Decaying $\epsilon$-greedy)
        if action chosen is primitve then
            Observe reward ($r_{t+1}$) and next state ($s_{t+1}$) by playing the action $a_t$
            $Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$
        end
        if action chosen is an option then
            Rollout the option's policy and observe rewards $(r_{t+1}, r_{t+2}, \dots r_{t+\tau})$
            $\bar{r}_{t+\tau} \leftarrow \mathrm{r}_{t+1} + \gamma r_{t+2} + \dots \gamma^{\tau-1} r_{t+\tau}$
              $Q(s_t, a_t) = Q(s_t, a_t) + \alpha(\bar{r}_{t+\tau} + \gamma^{\tau} \max_a Q(s_{t+\tau}, a) - Q(s_t, a_t))$
        end
        $s_t \leftarrow s_{t+1}$
    end
end
```

### 3.2 Implementation

Here is the code snippet implementing the SMDP Q learning algorithm 3.2.

```
action_option_list = ['down', 'up', 'right', 'left', 'pick', 'put', 'R',
↪  'G', 'Y', 'B']

goal = {'R': [0, 0], 'G': [0, 4], 'Y': [4, 0], 'B': [4, 3]}

```

```
np.random.seed(0)

# q table for SMDP and options
q_smdp = np.zeros((env.observation_space.n, 6))
q_option = {'R' : np.zeros((env.observation_space.n, env.action_space.n - 2)), 'G' : np.zeros((env.observation_space.n, env.action_space.n - 2)), 'Y' : np.zeros((env.observation_space.n, env.action_space.n - 2)), 'B' : np.zeros((env.observation_space.n, env.action_space.n - 2))}

# epsilon greedy parameters
epsilon_start = 1
epsilon_end = 0.0001
epsilon_decay = 0.99
eps = epsilon_start

episodic_rewards_list = []

for i in range(10000):
    state = env.reset()
    state = state[0]
    done = False
    episodic_reward = 0

    if i >= 9995:
        x, y, p, d = list(env.decode(state))
        print(f"Start State: {x, y, p, d}")
        print(env.render())

    while not done:

        action = epsilon_greedy(q_smdp[state], eps)
        eps = max(epsilon_end, eps * epsilon_decay)

        if i >= 9995:
            print("Action executed ", action_option_list[action + 4])

        # primitive actions - 'pick' and 'put'
        if action < 2:

            next_state, reward, done, terminated, _ = env.step(action + 4)
            done = done or terminated

            q_smdp[state, action] += alpha_smdp * (reward + gamma * np.max(q_smdp[next_state]) - q_smdp[state, action])
            state = next_state
            episodic_reward += reward
            if i >= 9995:
                print(env.render())

        # options - 'R', 'G', 'Y', 'B'
        else:
            option = action_option_list[action + 4]
            option_done = False
            option_reward = 0
            reward_bar = 0
            tau = 0
            state_start = state

            while not option_done:

                optact = epsilon_greedy(q_option[option][state], eps)

                next_state, reward, done, terminated, _ = env.step(optact)
                done = done or terminated
                x, y, _, _ = list(env.decode(next_state))
                if [x, y] == goal[option]:
                    option_done = True
                option_reward += reward
                # TD update
                q_option[option][state, optact] += alpha_td * (reward +
                ↪   gamma * np.max(q_option[option][next_state]) -
                ↪   q_option[option][state, optact])

                reward_bar = reward_bar + gamma**(tau) * reward
                tau += 1

                state = next_state
                episodic_reward += reward
                if i >= 9995:
                    print(env.render())
                if done:
                    break

            # SMDP update
            q_smdp[state_start, action] += alpha_smdp * (reward_bar +
            ↪   gamma**(tau) * np.max(q_smdp[state]) - q_smdp[state_start,
            ↪   action])

    episodic_rewards_list.append(episodic_reward)
    print("Episode {} Reward {}".format(i, episodic_reward))
```

Listing 1: Code snippet for implementing SMDP Q learning

### 3.3 Results

In this section, we present the results obtained from performing SMDP Q learning on the Taxi-v3 environment.

We consider $\gamma = 0.9$ (as given), learning rate $\alpha = 0.1$ and employ the epsilon greedy action selection policy to select an action from the Q values for a particular state for the learning option's policy as well as the agent's policy to learn actions/options. We also decay the epsilon at every step to ensure exploration at the beginning of training and exploitation towards the end (start $\epsilon = 1$, end $\epsilon = 0.0001$, $\epsilon$ decay $= 0.99$).

We perform SMDP Q Learning for 10000 episodes and plot the running average of the episodic reward of the last 100 episodes v/s the episodes.

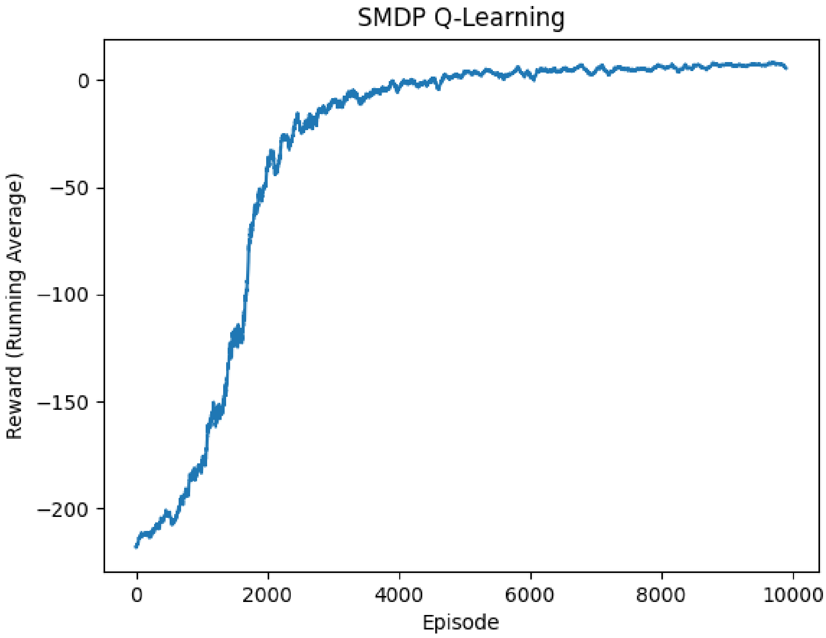
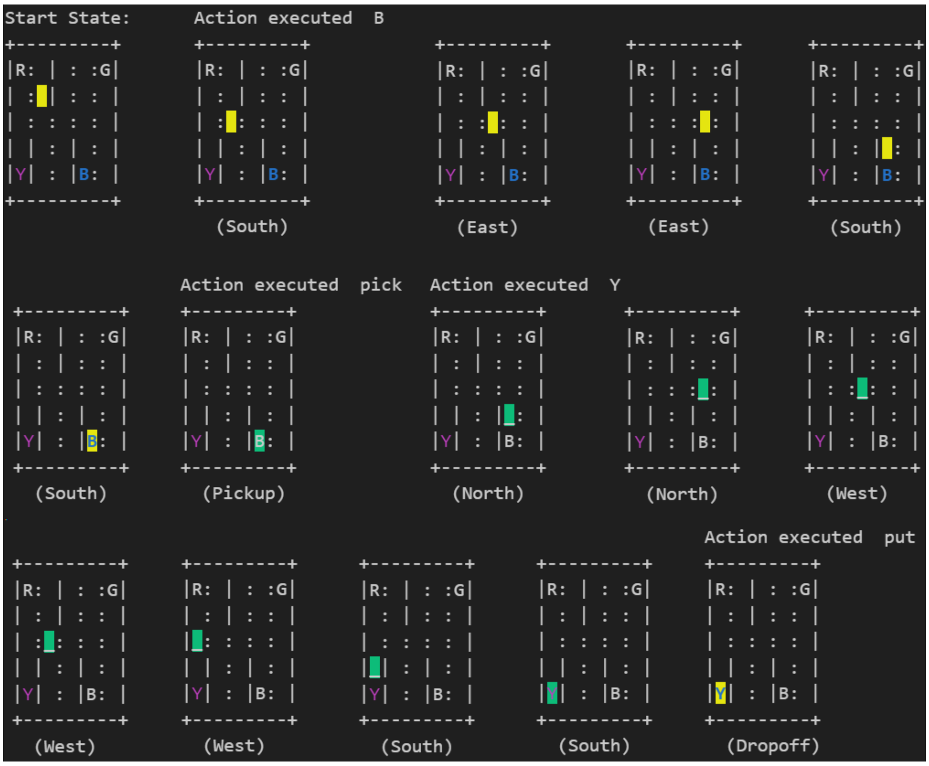


Figure 2: Running Average of Episodic Reward v/s Number of Episodes

From Figure 2, we observe that the agent learns to choose actions/options that optimize the episodic reward. This also suggests that every option's policy is being learned simultaneously. To further analyze the exact policy learned by the agent, we render the final episode as well as visualize the policy that is greedy with respect to the learned Q values of every state.

Here is the rendering of the $10000^{th}$ episode in Figure 3 to visualize the agent's policy.

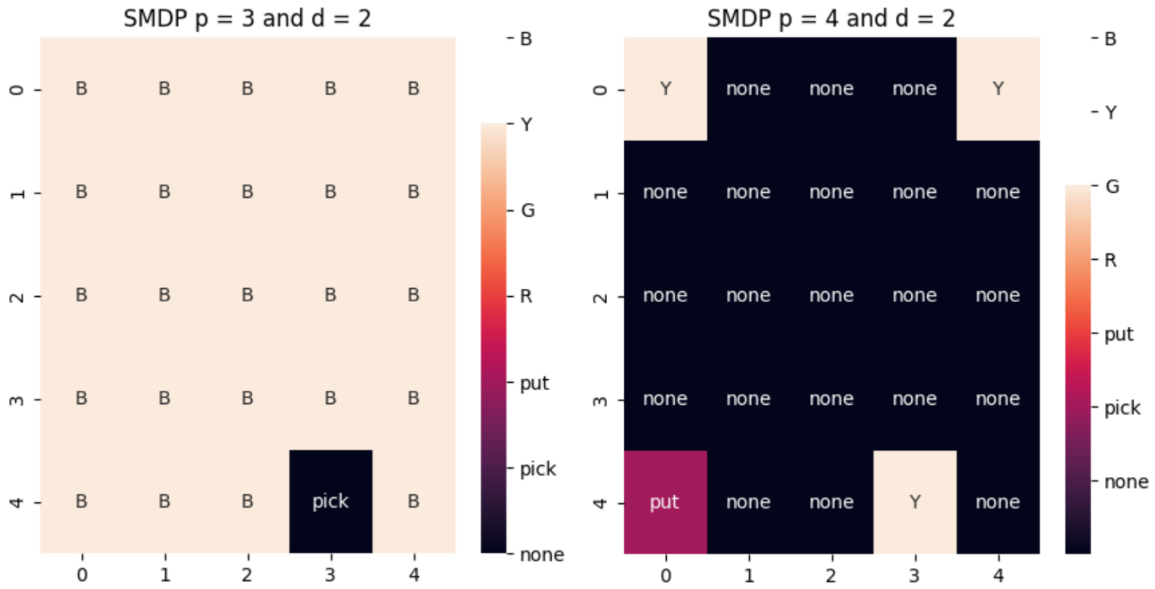


```
Start State:     Action executed  B
+---------+      +---------+      +---------+      +---------+      +---------+
|R: | : :G|      |R: | : :G|      |R: | : :G|      |R: | : :G|      |R: | : :G|
| :█| : : |      | : | : : |      | : | : : |      | : | : : |      | : | : : |
| : : : : |      | :█: : : |      | : :█: : |      | : : :█: |      | : : : : |
| | : | : |      | | : | : |      | | : | : |      | | : | : |      | | : |█: |
|Y| : |B: |      |Y| : |B: |      |Y| : |B: |      |Y| : |B: |      |Y| : |B: |
+---------+      +---------+      +---------+      +---------+      +---------+
                   (South)          (East)           (East)           (South)

                 Action executed  pick  Action executed  Y
+---------+      +---------+      +---------+      +---------+      +---------+
|R: | : :G|      |R: | : :G|      |R: | : :G|      |R: | : :G|      |R: | : :G|
| : | : : |      | : | : : |      | : | : : |      | : | : : |      | : | : : |
| : : : : |      | : : : : |      | : : : : |      | : : :█: |      | : :█: : |
| | : | : |      | | : | : |      | | : |█: |      | | : | : |      | | : | : |
|Y| : |B: |      |Y| : |B: |      |Y| : |B: |      |Y| : |B: |      |Y| : |B: |
+---------+      +---------+      +---------+      +---------+      +---------+
  (South)         (Pickup)          (North)          (North)          (West)

                                                                    Action executed  put
+---------+      +---------+      +---------+      +---------+      +---------+
|R: | : :G|      |R: | : :G|      |R: | : :G|      |R: | : :G|      |R: | : :G|
| : | : : |      | : | : : |      | : | : : |      | : | : : |      | : | : : |
| :█: : : |      |█: : : : |      | : : : : |      | : : : : |      | : : : : |
| | : | : |      | | : | : |      |█| : | : |      | | : | : |      | | : | : |
|Y| : |B: |      |Y| : |B: |      |Y| : |B: |      |Y| : |B: |      |Y| : |B: |
+---------+      +---------+      +---------+      +---------+      +---------+
  (West)           (West)           (South)          (South)         (Dropoff)
```

Figure 3: Rendering of environment when the start state is (1, 1, 3, 2). The reward for this episode is 7.

As we can see from the figure above, the start state is (1, 1, 3, 2), implying that the passenger is at 3 (Blue), drop location is 2 (Yellow) and the taxi starts at (1, 1). The agent correctly learns to first pick up the passenger from 3 in the shortest path possible by executing the 'B' option. It then executes the primitive action 'pick', followed by the option 'Y' to navigate to the drop off location in the shortest possible path. Once the taxi reaches Yellow, it correctly executes the 'put' option to successfully drop off the passenger in the shortest path possible.

We further plot some selective Q value tables used in the above rendered episode to visualize the learnt agent policy and option policies after 10000 episodes in Figures 4 and 5. Here, we have plotted the greedy Q policy to visualise the maximum Q actions for each location of the taxi on the grid for (p, d) = (3, 2) (pickup leg of the episode) and (4, 2) (drop off leg of the episode).

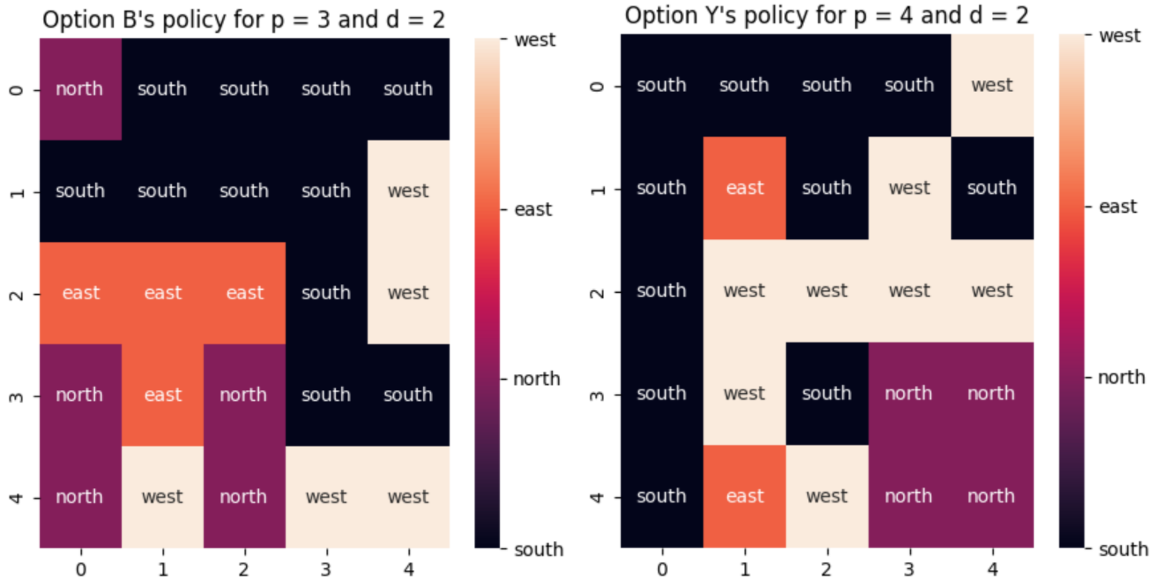


(a) Passenger: Blue, Destination: Yellow. (b) Passenger: Inside taxi, Destination: Yellow.

Figure 4: Visualizing Greedy Q policy for (p, d) = (3, 2) and (4, 2) respectively.

As we can see from Figure 4, all states (other than the state where the taxi is at Blue) have the highest Q value for the option ‘B’, which is the pick up location. The state where the taxi is at Blue has ‘pick’ as the highest Q value action, since it needs to pick up the passenger at this location.

After the pick up in the Blue location, we see that the agent has the highest Q value for the Y option in this state. This is as expected since now the agent needs to navigate to the drop location (Yellow). At the drop location, the ‘put’ option has the highest Q value, leading to the successful drop off of the passenger. We see several ’none’ states in this Q table. The ‘none’ option is outputted wherever the Q values of all actions in that state are zero, implying that the state was never visited. The states other than R, G, Y and B never get updated once the passenger is picked up, since the taxi immediately starts executing one of the options with the starting state at Blue, and the option does not terminate until it reaches the chosen option’s designated location. In SMDP Q learning, we only update the option value of the starting state, so the intermediate states are not updated.



(a) Greedy Q policy for Option B

(b) Greedy Q policy for Option Y

Figure 5: Visualising the greedy Q policy for the options followed in 3 (Option B for pick-up, Option Y for drop).

Figure 5 shows the greedy Q policies for the two options executed in the rendering in Figure 3, i.e. ‘B’ and ‘Y’. We see that the taxi correctly moves towards the Blue and Yellow locations respectively, while taking the shortest path and avoiding obstacles.

### 3.4 Inferences

- We see that the taxi is able to learn the optimal policies for both the agent as well as the options in 10000 episodes using SMDP Q Learning.
- In SMDP Q Learning, we have used the set of actions available to the agent as [‘pick’, ‘put’, ‘R’, ‘G’, ‘B’, ‘Y’]. When the agent chooses ‘pick’ or ‘put’, the Q value for these gets updated with the regular Q learning update.
- However, when the agent picks one of the 4 options, the agent starts to execute that option’s policy. There are no updates to the Q table for actions until the end of the option, only the option policy Q values get updated during this time.
- When the option is done, we update the Q value for the option with the cumulative discounted reward. This, in a way, is treating the option like a primitive action with a reward equal to the cumulative discounted reward received at the end of option completion.
- Thus, the agent will act in a way to maximize both the discounted reward for option Q value updation, as well as the overall reward for picking actions (defined in list above). This leads to the agent learning the optimal policy for each option, as well as the optimal policy for picking the options.
- Also, having the option’s policy being learned during training ensures that the path taken by the taxi when it chooses that particular option is optimized. This translates to the taxi traveling from any location to any designated location in the shortest path possible.

# 4 Intra Option Q Learning

The second algorithm we use in this assignment is Intra Option Q Learning. The pseudocode is provided in Algorithm 2.

## 4.1 Algorithm

**Algorithm 2:** Intra Option Q Learning

```
Initialize $\alpha$ (learning rate) and $\gamma$ (discount factor)
Initialize Q table
for every episode do
    while episode not done do
        Select action ($a_t$) or option ($o_t$) from state ($s_t$) using a policy derived from Q
          (e.g. Decaying $\epsilon$-greedy)
        if action chosen is primitve then
            Observe reward ($r_{t+1}$) and next state ($s_{t+1}$) by playing the action $a_t$
            $Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$
            for all the options that could take this action do
                if $s_{t+1}$ is not terminating for the option then
                    $Q(s_t, o) = Q(s_t, o) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, o) - Q(s_t, o))$
                end
                if $s_{t+1}$ is terminating for the option then
                    $Q(s_t, o) = Q(s_t, o) + \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, o))$
                end
            end
        end
        if action chosen is an option then
            while option is under execution do
                Choose the next action ($a$) from state ($s$) based on the option's policy
                Observe reward ($r$) and next state ($s'$) by playing the action $a$
                $Q(s, a) = Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$
                for all the options that could take this action do
                    if $s'$ is not terminating for the option then
                        $Q(s, o) = Q(s, o) + \alpha(r + \gamma Q(s', o) - Q(s, o))$
                    end
                    if $s'$ is terminating for the option then
                        $Q(s, o) = Q(s, o) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, o))$
                    end
                end
            end
        end
        $s_t \leftarrow s_{t+1}$
    end
end
```

## 4.2 Implementation

Here is the code snippet implementing the Intra Option Q learning algorithm 4.2.

```python
action_option_list = ['down', 'up', 'right', 'left', 'pick', 'put', 'R', 'G', 'Y', 'B']

goal = {'R': [0, 0], 'G': [0, 4], 'Y': [4, 0], 'B': [4, 3]}

np.random.seed(0)

# q table for intra option and options
q_ioql = np.zeros((env.observation_space.n, 6))
q_option_ioql = {'R' : np.zeros((env.observation_space.n, env.action_space.n - 2)), 'G' : np.zeros((env.observation_space.n, env.action_space.n - 2)), 'Y' : np.zeros((env.observation_space.n, env.action_space.n - 2)), 'B' : np.zeros((env.observation_space.n, env.action_space.n - 2))}

# parameters
gamma_ioql = 0.9
gamma_policy = 0.9
alpha_ioql = 0.1

# epsilon greedy parameters
epsilon_ioql = 1
epsilon_opt = 1
episodic_rewards_list_ioql = []
eps_min = 0.0001
eps_decay = 0.99

for i in range(10000):
    state = env.reset()
    state = state[0]
    done = False
    episodic_reward = 0

    if i >= 9995:
        x, y, p, d = list(env.decode(state))
        print(x, y, p, d)
        print("Start State")

    while not done:
        epsilon_ioql = max(eps_min,eps_decay*epsilon_ioql)
        epsilon_opt = max(eps_min,eps_decay*epsilon_opt)

        action = epsilon_greedy(q_ioql[state], epsilon_ioql)

        if i >=9995:
            print("Action executed", action_option_list[action + 4])

        # primitive actions
        if action < 2:

            next_state, reward, done, terminated, _= env.step(action + 4)
            done = done or terminated
            x, y, _, _ = list(env.decode(next_state))
            q_ioql[state, action] += alpha_ioql * (reward + gamma_ioql * np.max(q_ioql[next_state]) - q_ioql[state, action])

            state = next_state
            episodic_reward += reward
            if i >= 9995:
                print(env.render())

        # options - 'R', 'G', 'Y', 'B'
        else:
            option = action_option_list[action + 4]
            init_state = state
            option_done = False

            while (option_done == False):

                optact = epsilon_greedy(q_option_ioql[option][state], epsilon_opt)
                next_state, reward, done, terminated, _ = env.step(optact)
                done = done or terminated
                x, y, _, _ = list(env.decode(next_state))
                if [x, y] == goal[option]:
                    option_done = True

                for idx in [2, 3, 4, 5]:
                    opt = action_option_list[idx + 4]

                    # intra option Q learning update
                    if [x, y] == goal[opt]:
                        q_option_ioql[opt][state, optact] += alpha_td * (reward + gamma_policy * np.max(q_option_ioql[opt][next_state]) - q_option_ioql[opt][state, optact])
                        q_ioql[state, idx] += alpha_ioql * (reward + gamma_ioql * np.max(q_ioql[next_state]) - q_ioql[state, idx])
                    else:
                        q_option_ioql[opt][state, optact] += alpha_td * (reward + gamma_policy * np.max(q_option_ioql[opt][next_state]) - q_option_ioql[opt][state, optact])
                        q_ioql[state, idx] += alpha_ioql * (reward + gamma_ioql * (q_ioql[next_state][idx]) - q_ioql[state, idx])

                state = next_state
                episodic_reward += reward
                if i >= 9995:
                    print(env.render())
                if done:
                    break

    episodic_rewards_list_ioql.append(episodic_reward)
    print(f"Episode {i}: Reward = {episodic_reward}")
```

Listing 2: Code snippet for implementing Intra Option Q learning

### 4.3 Results

In this section, we present the results obtained from performing Intra Option Q Learning on the Taxi-v3 environment.

We consider $\gamma = 0.9$ (as given), learning rate $\alpha = 0.1$ and employ the epsilon greedy action selection policy to select an action from the Q values for a particular state for the learning option's policy as well as the agent's policy to learn actions/options. We also decay the epsilon at every step to ensure exploration at the beginning of training and exploitation towards the end (start $\epsilon = 1$, end $\epsilon = 0.0001$, $\epsilon$ decay $= 0.99$).

We perform Intra Option Q Learning for 10000 episodes and plot the running average of the episodic reward of the last 100 episodes v/s the episodes.

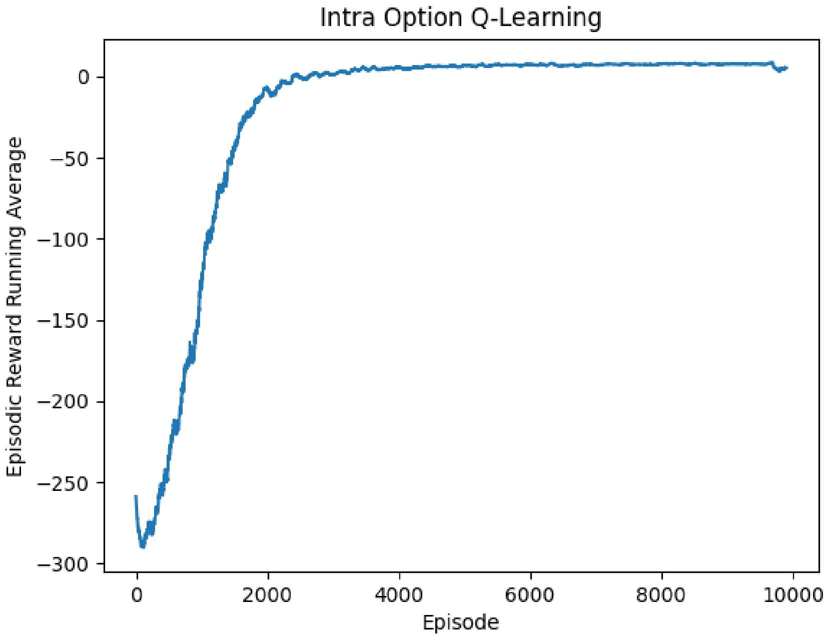
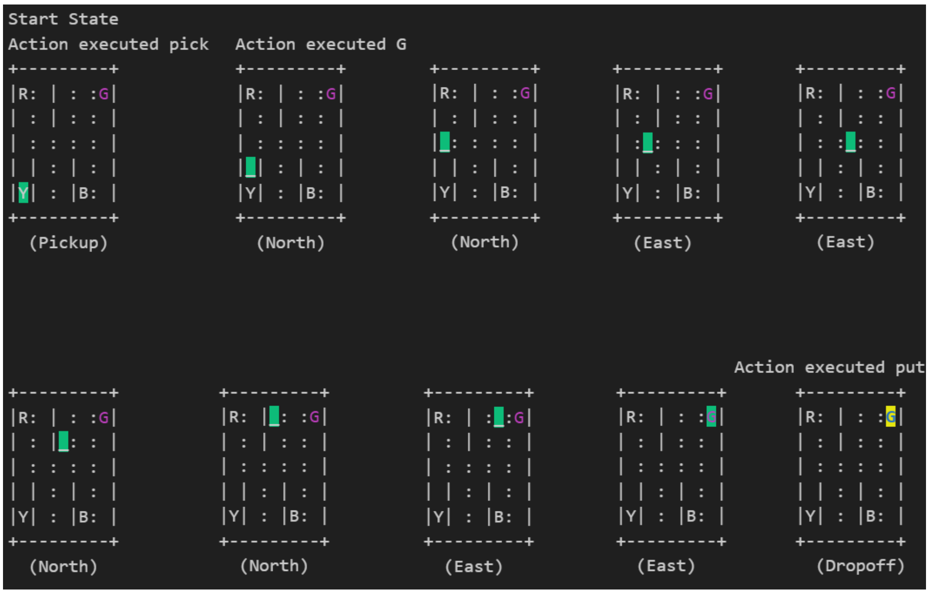


Figure 6: Running Average of Episodic Reward v/s Number of Episodes

From Figure 6, we observe that the agent learns to choose actions/options that optimize the episodic reward. This also suggests that every option's policy is being learned simultaneously. To further analyze the exact policy learned by the agent, we render the final episode as well as visualize the policy that is greedy with respect to the learned Q values of every state.

Here is the rendering of the $10000^{th}$ episode in Figure 7 to visualize the agent's policy.

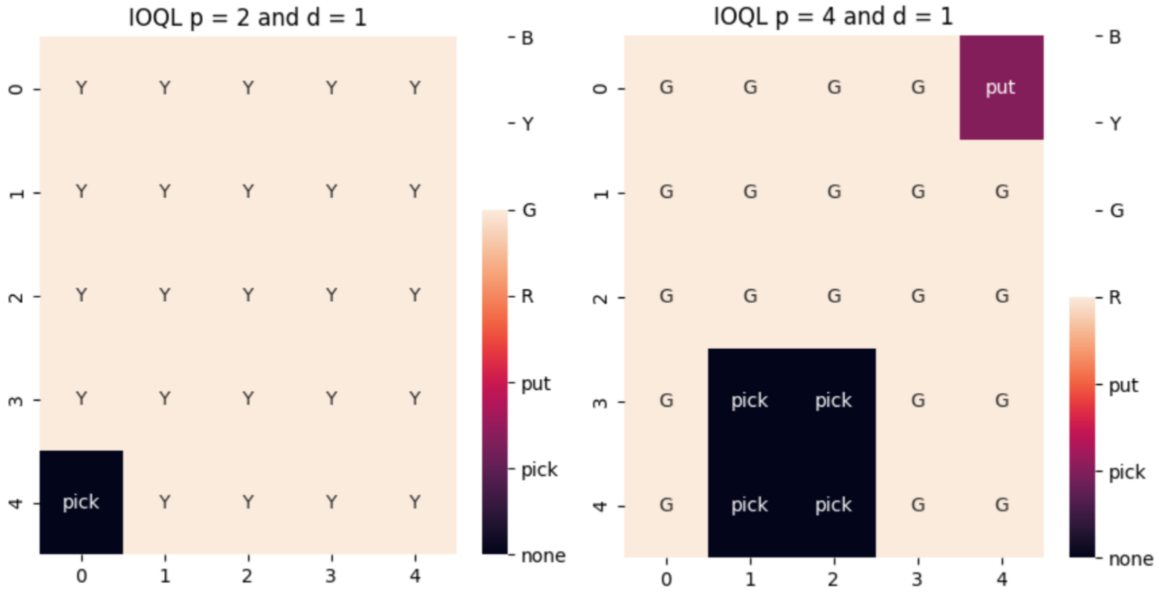


```
Start State
Action executed pick   Action executed G
+---------+            +---------+        +---------+       +---------+       +---------+
|R: | : :G|            |R: | : :G|        |R: | : :G|       |R: | : :G|       |R: | : :G|
| : | : : |            | : | : : |        | : | : : |       | : | : : |       | : | : : |
| : : : : |            | : : : : |        | : : : : |       | : : : : |       | : : : : |
| | : | : |            | | : | : |        | | : | : |       | | : | : |       | | : | : |
|Y| : |B: |            |Y| : |B: |        |Y| : |B: |       |Y| : |B: |       |Y| : |B: |
+---------+            +---------+        +---------+       +---------+       +---------+
  (Pickup)               (North)            (North)           (East)            (East)




                                                                        Action executed put
+---------+          +---------+         +---------+        +---------+       +---------+
|R: | : :G|          |R: | : :G|         |R: | : :G|        |R: | : :G|       |R: | : :G|
| : | : : |          | : | : : |         | : | : : |        | : | : : |       | : | : : |
| : : : : |          | : : : : |         | : : : : |        | : : : : |       | : : : : |
| | : | : |          | | : | : |         | | : | : |        | | : | : |       | | : | : |
|Y| : |B: |          |Y| : |B: |         |Y| : |B: |        |Y| : |B: |       |Y| : |B: |
+---------+          +---------+         +---------+        +---------+       +---------+
  (North)              (North)             (East)             (East)            (Dropoff)
```

Figure 7: Rendering of the environment when the start state is (4, 0, 2, 1). The reward for this episode is 11.

As we can see from the figure above, the start state is (4, 0, 2, 1), implying that the passenger is at 2 (Yellow), the drop location is 1 (Green) and the taxi starts at (4, 0), which happens to be the same location as the passenger. The agent correctly learns to first pick up the passenger at Y by executing the primitive action 'pick', followed by the option 'G' to navigate to the drop-off location in the shortest possible path. Once the taxi reaches Green, it correctly executes the 'put' option to successfully drop off the passenger in the shortest path possible.

We further plot some selective Q value tables used in the above rendered episode to visualize the learned agent policy and option policies after 10000 episodes in Figures 8 and 9. Here, we have plotted the greedy Q policy to visualize the maximum Q actions for each location of the taxi on the grid for (p, d) = (2, 1) (pickup leg of the episode) and (4, 1) (drop off leg of the episode).



(a) Passenger: Yellow, Destination: Green.

(b) Passenger: Inside taxi, Destination: Green.

Figure 8: Visualizing Greedy Q policy for (p, d) = (2, 1) and (4, 1) respectively.

As we can see from Figure 8, all states (other than the state where the taxi is at Yellow) have the highest Q value for the option ‘Y’, which is the pick up location. The state where the taxi is at Yellow has ‘pick’ as the highest Q value action, since it needs to pick up the passenger at this location.

After the pick up in the Yellow location, we see that the agent has the highest Q value for the G option in this pick up state. This is as expected since now the agent needs to navigate to the drop location (Green). At the drop location, the ‘put’ action has the highest Q value, leading to the successful drop off of the passenger. We see some ’pick’ values in this Q table even after picking up the passenger. This may be because Intra Option Q Learning updates every state and option even while executing a different option, unlike SMDP. However, since the agent (greedily) picks option G immediately after pick up and this option does not end until the agent reaches the end location (Green), so the policy is still accurate and drops the passenger off successfully.

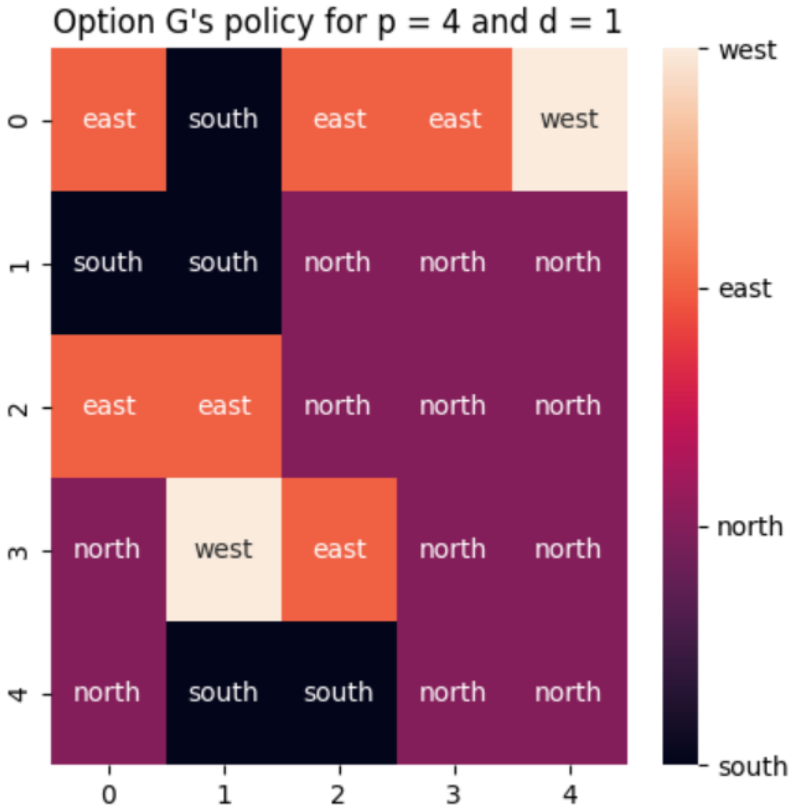


Figure 9: Visualising the greedy Q policy for Option G followed in 7 for drop off.

Figure 9 shows the greedy Q policies for Option G executed for drop off in the rendering in Figure 7 (pick up in this case is trivial, since taxi spawns at pick up location). We see that the taxi correctly moves towards the location Green from the pick up location, while taking the shortest path and avoiding obstacles.

## 4.4 Inferences

- We see that the taxi is able to learn the optimal policies for both the agent as well as the options in 10000 episodes using Intra Option Q Learning.
- In Intra Option Q Learning also we have used the set of actions available to the agent as [‘pick’, ‘put’, ‘R’, ‘G’, ‘B’, ‘Y’].
- When the agent chooses ‘pick’ or ‘put’, the Q value for these gets updated with the regular Q learning update (since our option policy has only ‘down’, ‘up’, ‘left’ and ‘right’ as possible actions, there is no need for extra updates to the option Q values for ‘pick’ and ‘put’).
- However, when the agent picks one of the 4 options, the agent starts to execute that option’s policy. During option execution, we update the option’s policy as well as the Q value for every option that allows the particular state-action pair being executed at every step until option completion.
- Thus, Intra option Q learning attempts to use each step of option execution and every single state-action pair encountered to simultaneously update and learn about all options that are consistent with that step. This simultaneous update increases

sample efficiency, speeds up learning and converges faster to the optimal policy than SMDP Q learning, as seen from Figure 18.

# 5 Alternate Options

We consider the options ‘go to passenger’ and ‘go to destination’ which are mutually exclusive to the options considered above (‘R’, ‘G’, ‘Y’ and ‘G’).

Previously, options like ‘R’, ‘G’, ‘Y’, and ‘B’ represented fixed destinations for passengers. We learned separate policies for each option that allow the taxi to navigate. Those options end upon reaching the designated location.

The new options, ‘go to passenger’ and ‘go to destination’, leverage the state information that includes the passenger’s location and final destination. The ‘go to passenger’ option terminates when the taxi reaches the passenger’s initial location, while ‘go to destination’ finishes upon arrival at the passenger’s final stop.

The set of actions that the agent can choose from any given state includes the ‘pick’ and ‘put’ actions along with ‘go to passenger’ and ‘go to destination’ options.

Similar to the experiments performed above, the policies for ‘go to passenger’ and ‘go to destination’ are learned with Q-learning, while the overall agent policy utilizes SMDP Q-learning and Intra Option Q-learning.

## 5.1 SMDP Q Learning

We consider $\gamma = 0.9$ (as given), learning rate $\alpha = 0.1$ and employ the epsilon greedy action selection policy to select an action from the Q values for a particular state for the learning option’s policy as well as the agent’s policy to learn actions/options. We also decay the epsilon at every step to ensure exploration at the beginning of training and exploitation towards the end (start $\epsilon = 1$, end $\epsilon = 0.0001$, $\epsilon$ decay $= 0.99$).

We perform SMDP Q Learning for 10000 episodes and plot the running average of the episodic reward of the last 100 episodes v/s the episodes.

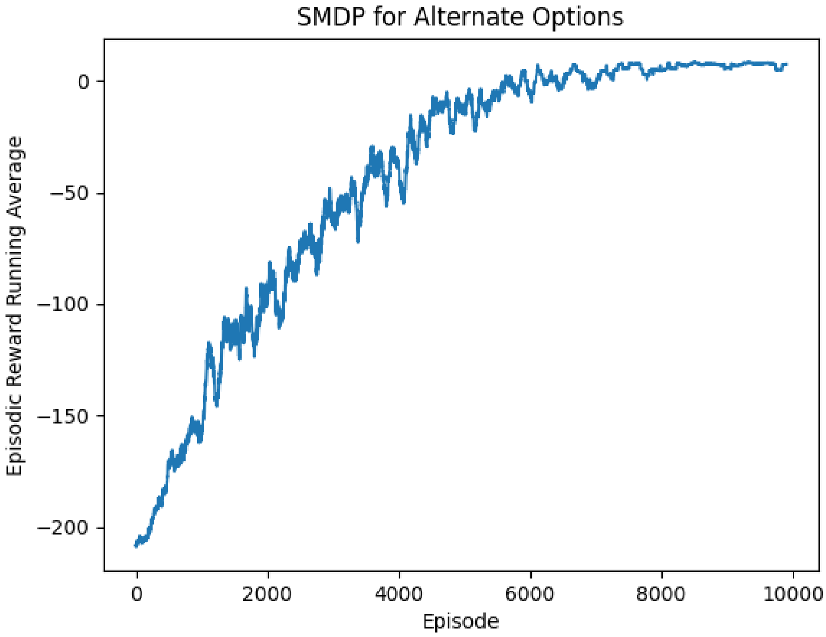
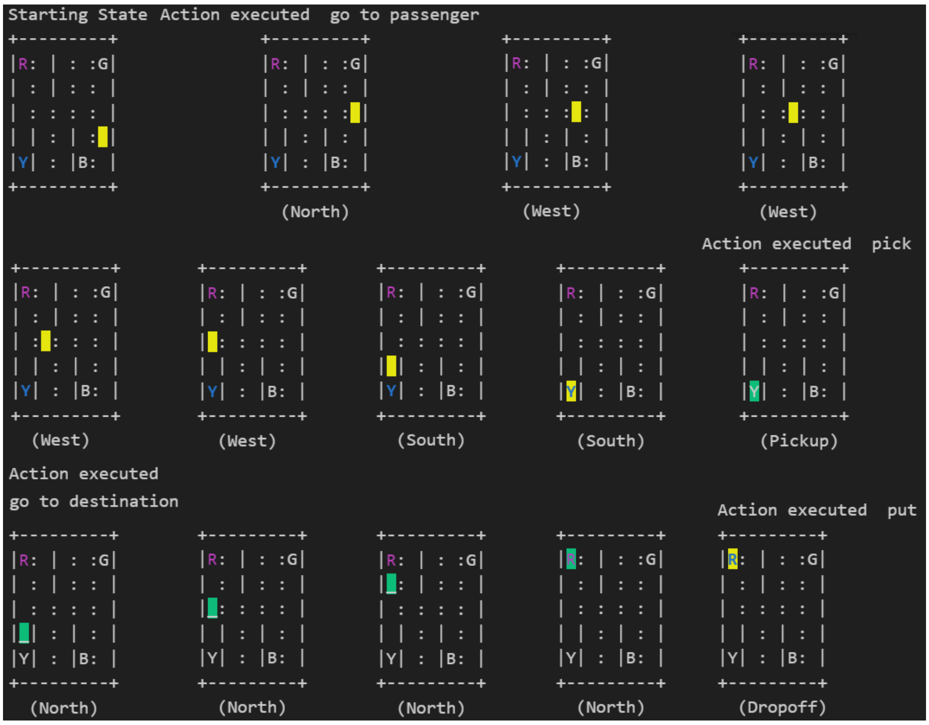


Figure 10: Running Average of Episodic Reward v/s Number of Episodes

From Figure 10, we observe that the agent learns to choose actions/options that optimize the episodic reward. This also suggests that every option's policy is being learned simultaneously. To further analyze the exact policy learned by the agent, we render the final episode as well as visualize the policy that is greedy with respect to the learned Q values of every state.

Here is the rendering of the $10000^{th}$ episode in Figure 11 to visualize the agent's policy.

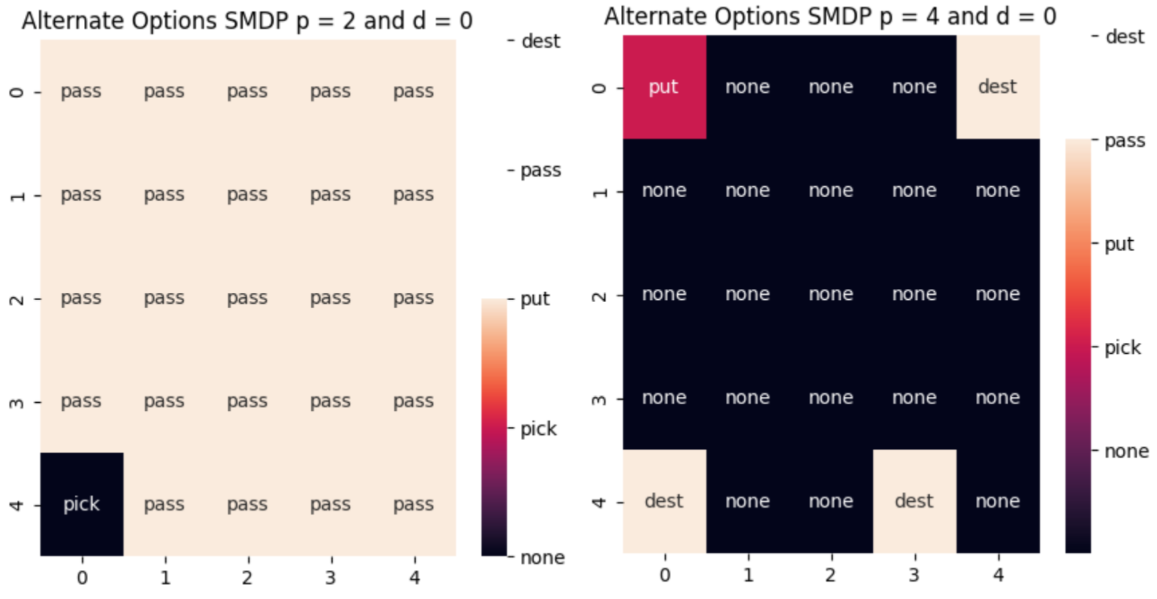


```
Starting State  Action executed  go to passenger
+---------+         +---------+          +---------+          +---------+
|R: | : :G|         |R: | : :G|          |R: | : :G|          |R: | : :G|
| : | : : |         | : | : : |          | : | : : |          | : | : : |
| : : : : |         | : : : :█|          | : : :█: |          | : :█: : |
| | : | :█|         | | : | : |          | | : | : |          | | : | : |
|Y| : |B: |         |Y| : |B: |          |Y| : |B: |          |Y| : |B: |
+---------+         +---------+          +---------+          +---------+
                      (North)              (West)               (West)
                                                         Action executed  pick
+---------+    +---------+    +---------+    +---------+    +---------+
|R: | : :G|    |R: | : :G|    |R: | : :G|    |R: | : :G|    |R: | : :G|
| : | : : |    | : | : : |    | : | : : |    | : | : : |    | : | : : |
| :█: : : |    |█: : : : |    | : : : : |    | : : : : |    | : : : : |
| | : | : |    | | : | : |    |█| : | : |    | | : | : |    | | : | : |
|Y| : |B: |    |Y| : |B: |    |Y| : |B: |    |Y| : |B: |    |Y| : |B: |
+---------+    +---------+    +---------+    +---------+    +---------+
  (West)         (West)         (South)        (South)        (Pickup)
Action executed
go to destination                                          Action executed  put
+---------+    +---------+    +---------+    +---------+    +---------+
|R: | : :G|    |R: | : :G|    |R: | : :G|    |R: | : :G|    |R: | : :G|
| : | : : |    | : | : : |    |█: | : : |    | : | : : |    | : | : : |
| : : : : |    |█: : : : |    | : : : : |    | : : : : |    | : : : : |
|█| : | : |    | | : | : |    | | : | : |    | | : | : |    | | : | : |
|Y| : |B: |    |Y| : |B: |    |Y| : |B: |    |Y| : |B: |    |Y| : |B: |
+---------+    +---------+    +---------+    +---------+    +---------+
  (North)        (North)        (North)        (North)       (Dropoff)
```

Figure 11: Rendering of the environment when the start state is (3, 4, 2, 0). The reward for this episode is 8.

As we can see from the figure above, the start state is (3, 4, 2, 0), implying that the passenger is at 2 (Yellow), drop location is 0 (Red) and the taxi starts at (3, 4). The agent correctly learns to first pick up the passenger from 2 (Yellow) in the shortest path possible by executing the 'go to passenger' option. It then executes the primitive action 'pick', followed by the option 'go to destination' to navigate to the drop-off location in the shortest possible path. Once the taxi reaches Red, it correctly executes the 'put' option to successfully drop off the passenger in the shortest path possible.

We further plot some selective Q value tables used in the above rendered episode to visualize the learned agent policy and option policies after 10000 episodes in Figures 12 and 13. Here, we have plotted the greedy Q policy to visualize the maximum Q actions for each location of the taxi on the grid for (p, d) = (2, 0) (pickup leg of the episode) and (4, 0) (drop off leg of the episode).



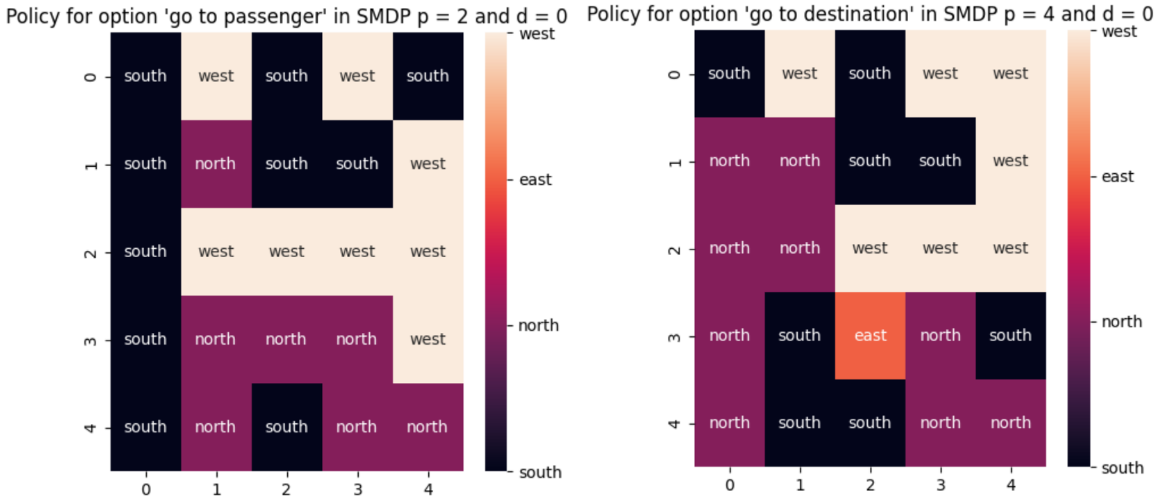
(a) Passenger: Yellow, Destination: Red.

(b) Passenger: Inside taxi, Destination: Red.

Figure 12: Visualizing Greedy Q policy for (p, d) = (2, 0) and (4, 0) respectively.

As we can see from Figure 12, all states (other than the state where the taxi is at Yellow) have the highest Q value for the option ‘go to passenger’ (represented in the figure as ‘pass’) The state where the taxi is at Yellow has ‘pick’ as the highest Q value action, since it needs to pick up the passenger at this location.

After the pick up in the Yellow location, we see that the agent has the highest Q value for the ‘go to passenger’ option in this state. This is as expected since now the agent needs to navigate to the drop location (Red). At the drop location, the ‘put’ option has the highest Q value, leading to the successful drop off of the passenger. We see several ’none’ states in this Q table. The ‘none’ option is outputted wherever the Q values of all actions in that state are zero, implying that the state was never visited. The states other than R, G, Y and B never get updated once the passenger is picked up, since the taxi immediately starts executing one of the options with the starting state at Yellow, and the option does not terminate until it reaches the chosen option’s designated location. In SMDP Q learning, we only update the option value of the starting state, so the intermediate states are not updated.



(a) Greedy Q policy for Option Go to Passenger (b) Greedy Q policy for Option Go to Destination

Figure 13: Visualising the greedy Q policy for the options followed in 11 (Option 'go to passenger' for pick-up, Option 'go to destination' for drop).

Figure 13 shows the greedy Q policies for the two options executed in the rendering in Figure 11, i.e. 'go to passenger' and 'go to destination'. We see that the taxi correctly moves towards the Yellow and Red locations respectively, while taking the shortest path and avoiding obstacles.

Thus, the taxi is able to learn the optimal policies for both the agent as well as the options in 10000 episodes.

## 5.2 Intra Option Q Learning

We consider $\gamma = 0.9$ (as given), learning rate $\alpha = 0.1$ and employ the epsilon greedy action selection policy to select an action from the Q values for a particular state for the learning option's policy as well as the agent's policy to learn actions/options. We also decay the epsilon at every step to ensure exploration at the beginning of training and exploitation towards the end (start $\epsilon = 1$, end $\epsilon = 0.0001$, $\epsilon$ decay $= 0.99$).

We perform Intra Option Q Learning for 10000 episodes and plot the running average of the episodic reward of the last 100 episodes v/s the episodes.

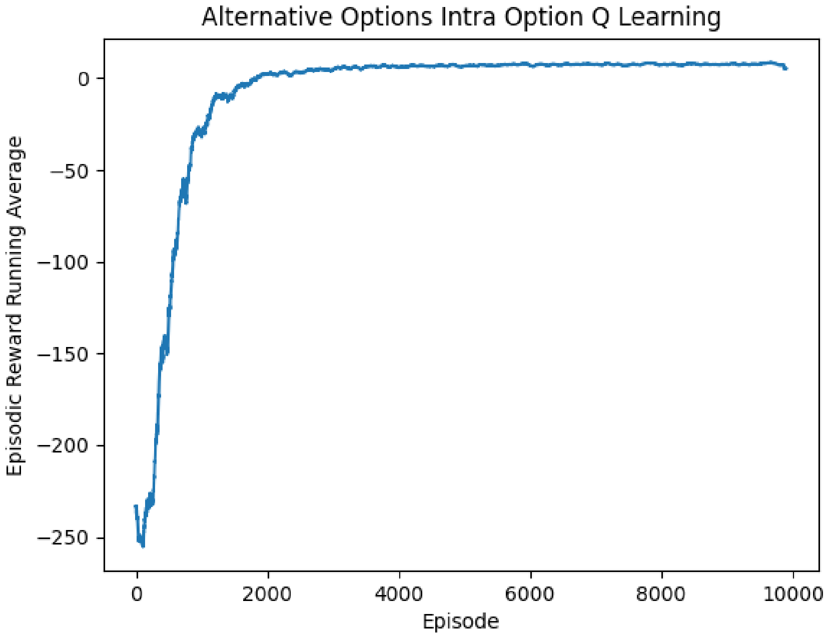
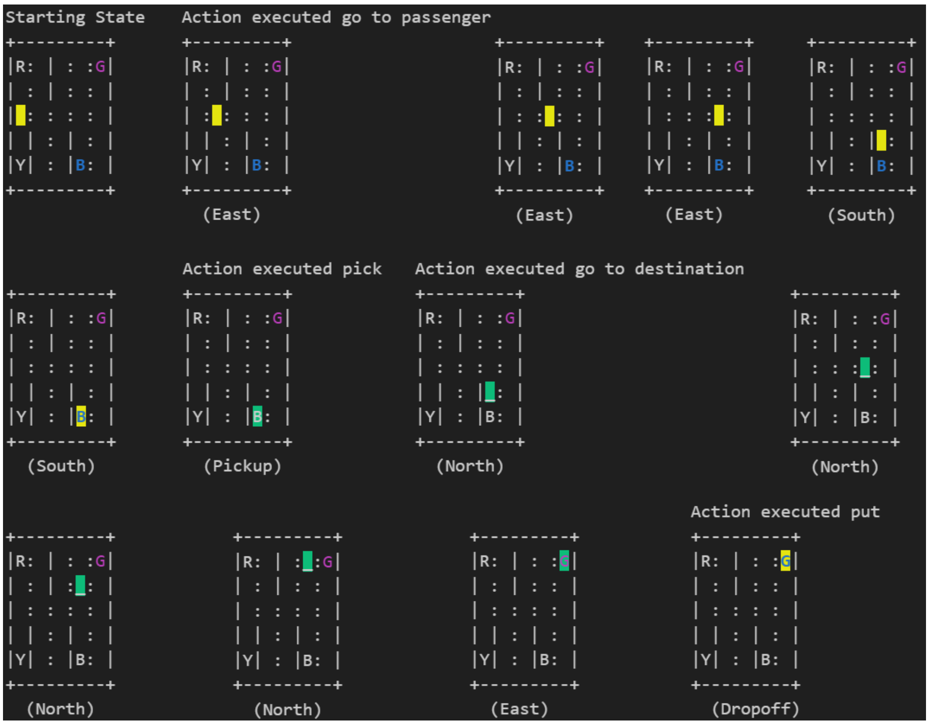


Figure 14: Running Average of Episodic Reward v/s Number of Episodes

From Figure 14, we observe that the agent learns to choose actions/options that optimize the episodic reward. This also suggests that every option's policy is being learned simultaneously. To further analyze the exact policy learned by the agent, we render the final episode as well as visualize the policy that is greedy with respect to the learned Q values of every state.

Here is the rendering of the $10000^{th}$ episode in Figure 15 to visualize the agent's policy.

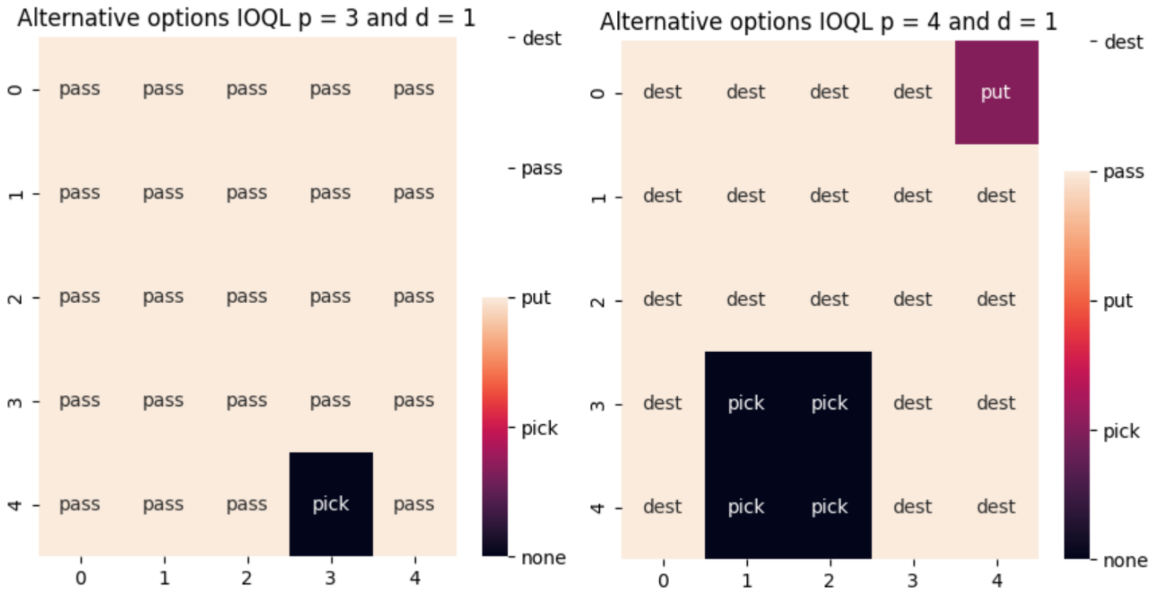


```
Starting State    Action executed go to passenger
+---------+       +---------+                    +---------+    +---------+     +---------+
|R: | : :G|       |R: | : :G|                    |R: | : :G|    |R: | : :G|     |R: | : :G|
| : | : : |       | : | : : |                    | : | : : |    | : | : : |     | : | : : |
|█: : : : |       | :█: : : |                    | : :█: : |    | : : :█: |     | : : : : |
| | : | : |       | | : | : |                    | | : | : |    | | : | : |     | | : |█: |
|Y| : |B: |       |Y| : |B: |                    |Y| : |B: |    |Y| : |B: |     |Y| : |B: |
+---------+       +---------+                    +---------+    +---------+     +---------+
                    (East)                         (East)         (East)          (South)

                  Action executed pick   Action executed go to destination
+---------+       +---------+            +---------+                           +---------+
|R: | : :G|       |R: | : :G|            |R: | : :G|                           |R: | : :G|
| : | : : |       | : | : : |            | : | : : |                           | : | : : |
| : : : : |       | : : : : |            | : : : : |                           | : : :█: |
| | : | : |       | | : | : |            | | : |█: |                           | | : | : |
|Y| : |B: |       |Y| : |B: |            |Y| : |B: |                           |Y| : |B: |
+---------+       +---------+            +---------+                           +---------+
  (South)           (Pickup)               (North)                               (North)

                                                                     Action executed put
+---------+            +---------+             +---------+           +---------+
|R: | : :G|            |R: | :█:G|             |R: | : :G|           |R: | : :G|
| : | :█: |            | : | : : |             | : | : : |           | : | : : |
| : : : : |            | : : : : |             | : : : : |           | : : : : |
| | : | : |            | | : | : |             | | : | : |           | | : | : |
|Y| : |B: |            |Y| : |B: |             |Y| : |B: |           |Y| : |B: |
+---------+            +---------+             +---------+           +---------+
  (North)                (North)                 (East)                (Dropoff)
```

Figure 15: Rendering of the environment when the start state is (2, 0, 3, 1). The reward for this episode is 9.

As we can see from the figure above, the start state is (2, 0, 3, 1), implying that the passenger is at 3 (Blue), the drop location is 1 (Green) and the taxi starts at (2, 0). The agent correctly learns to first pick up the passenger from 3 (Blue) in the shortest path possible by executing the 'go to passenger' option. It then executes the primitive action 'pick', followed by the option 'go to destination' to navigate to the drop-off location in the shortest possible path. Once the taxi reaches Green, it correctly executes the 'put' option to successfully drop off the passenger in the shortest path possible.

We further plot some selective Q value tables used in the above rendered episode to visualize the learned agent policy and option policies after 10000 episodes in Figures 16 and 17. Here, we have plotted the greedy Q policy to visualize the maximum Q actions for each location of the taxi on the grid for (p, d) = (3, 1) (pickup leg of the episode) and (4, 1) (drop off leg of the episode).



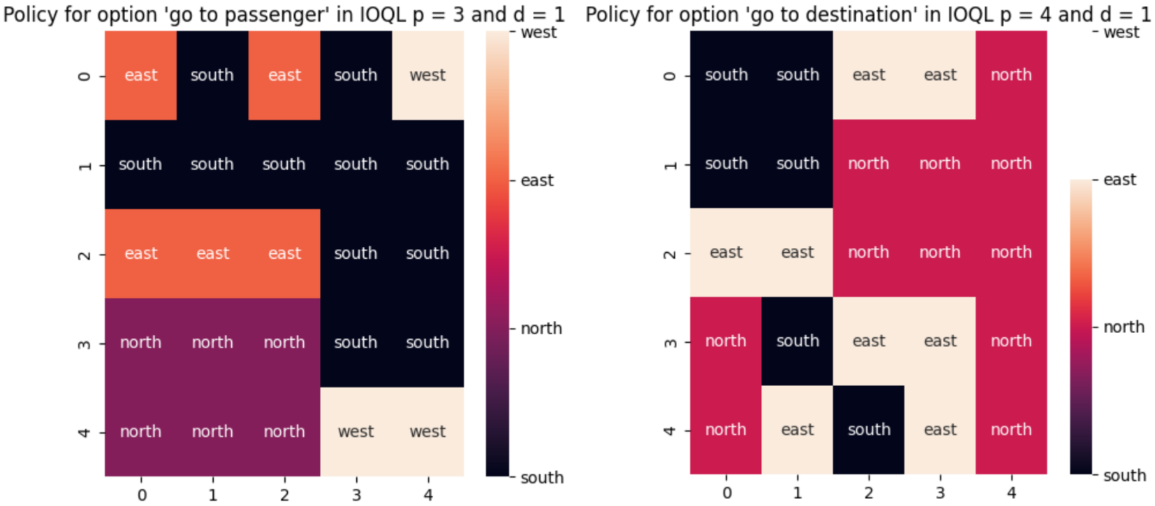
(a) Passenger: Blue, Destination: Green.

(b) Passenger: Inside taxi, Destination: Green.

Figure 16: Visualizing Greedy Q policy for (p, d) = (3, 1) and (4, 1) respectively.

As we can see from Figure 16, all states (other than the state where the taxi is at Blue) have the highest Q value for the option 'go to passenger' (represented with 'pass' in the figure). The state where the taxi is at Blue has 'pick' as the highest Q value action, since it needs to pick up the passenger at this location.

After the pick up in the Blue location, we see that the agent has the highest Q value for the G option in this pick up state. This is as expected since now the agent needs to navigate to the drop location (Green). At the drop location, the 'put' action has the highest Q value, leading to the successful drop off of the passenger. We see some 'pick' values in this Q table even after picking up the passenger. This may be because Intra Option Q Learning updates every state and option even while executing a different option, unlike SMDP. However, since the agent (greedily) picks option 'go to destination' immediately after pick up and this option does not end until the agent reaches the end location (Green), so the policy is still accurate and drops the passenger off successfully.


Policy for option 'go to passenger' in IOQL p = 3 and d = 1

|  | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | east | south | east | south | west |
| 1 | south | south | south | south | south |
| 2 | east | east | east | south | south |
| 3 | north | north | north | south | south |
| 4 | north | north | north | west | west |

Policy for option 'go to destination' in IOQL p = 4 and d = 1

|  | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | south | south | east | east | north |
| 1 | south | south | north | north | north |
| 2 | east | east | north | north | north |
| 3 | north | south | east | east | north |
| 4 | north | east | south | east | north |

(a) Greedy Q policy for Option Go to Passenger (b) Greedy Q policy for Option Go to Destination

Figure 17: Visualising the greedy Q policy for the options followed in 15 (Option 'go to passenger' for pick-up, Option 'go to destination' for drop).

Figure 17 shows the greedy Q policies for the two options executed in the rendering in Figure 15, i.e. 'go to passenger' and 'go to destination'. We see that the taxi correctly moves towards the Blue and Green locations respectively, while taking the shortest path and avoiding obstacles.

Thus, the taxi is able to learn the optimal policies for both the agent as well as the options in 10000 episodes.

### 5.3 Inferences

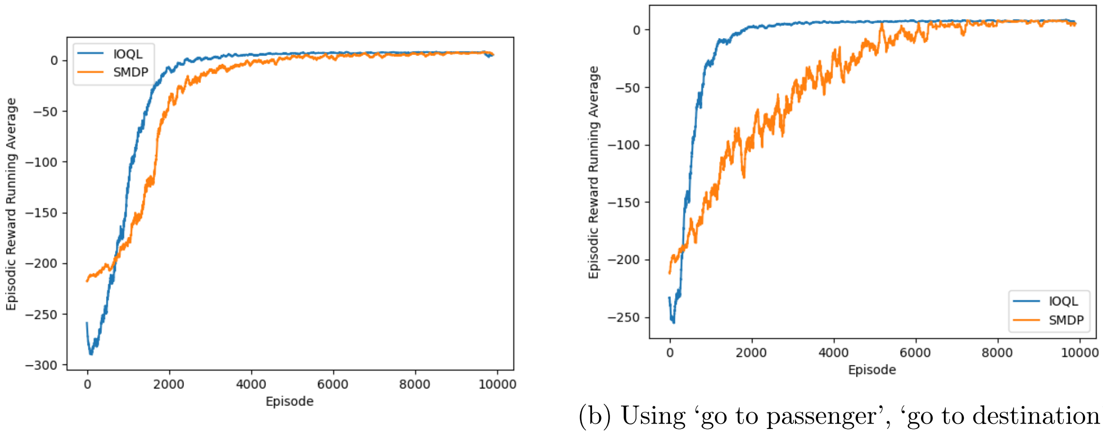
- From Figure 10 and 14 we observe that the agent successfully learns to the optimal policy among the set of [''pick', 'put', 'go to passenger' and 'go to destination'] actions and options in 10000 episodes.

- Similar to the experiments performed with options ['R', 'G', 'Y' and 'B'], the option's policy is also learned during training along with the agent's policy. Thus, given a state, the policy learnt for both 'go to passenger' option and 'go to destination' option is optimal.

- From Figure 18b, we observe that Intra Option Q learning learns faster when compared with SMDP Q learning algorithm.

- Also from Figure 18, upon close observation, we can infer that using the options ['go to passenger' and 'go to destination'] slows down learning compared to the options ['R', 'G', 'Y' and 'B'] when SMDP Q learning algorithm is being employed, whereas the opposite is true when Intra Option Q learning is used (i.e. using ['go to passenger' and 'go to destination'] speeds up learning as compared to the options ['R', 'G', 'Y' and 'B']).

# 6 Comparison between SMDP and Intra Option Q Learning



(a) Using ‘R’, ‘G’, ‘Y’, ‘B’ options

(b) Using ‘go to passenger’, ‘go to destination’ options

Figure 18: Comparative Running Average Reward Curves

- As expected, we observe that Intra Option Q learning algorithm learns faster than SMDP Q learning algorithm for both sets of options ([‘R’, ‘G’, ‘Y’ and ‘B’] and [‘go to passenger’ and ‘go to destination’]).

- This is expected because SMDP Q learning updates the Q table less frequently when compared to Intra Option Q learning.

- We observe that the SMDP graph has a higher variance than the Intra Option graph, especially in the case of the alternate options.

- We also observe that the Intra Option Q learning curve dips in the first few episodes of training. The SMDP Q learning curve has a higher reward than the Intra Option curve for the first few episodes.

- In the case of SMDP Q learning, the Q value for a state and primitive action pair is updated only when that pair is played in the environment. Similarly, when a state and option pair is played, the Q value for this pair is updated only once the option is terminated, and this update utilizes the cumulative discounted reward for updating.

- In the case of Intra Option Q learning, the Q value is simultaneously updated for all (state, action) pairs as well as all the (state, option) pairs for which the current action can be played, both when a primitive action is played as well as during every step of option execution.

# 7 GitHub Link

The link to the GitHub repository with the report, code, reward plots and other figures - Link.