Programming Assignment 3
CS6700

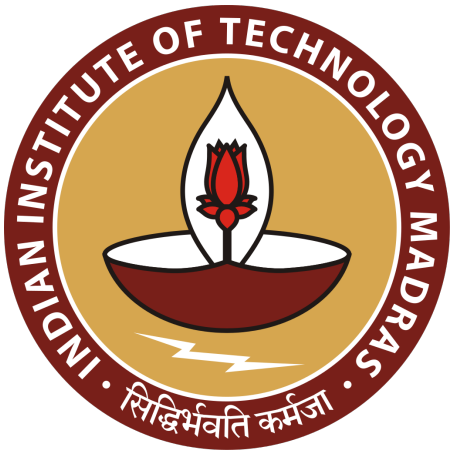
Hierarchical Reinforcement Learning

April 20, 2024

Balakumar R ME20B043

Girish Madhavan V ME20B072

Project Code Repo



Contents

1 Environment - `Taxi-v3`

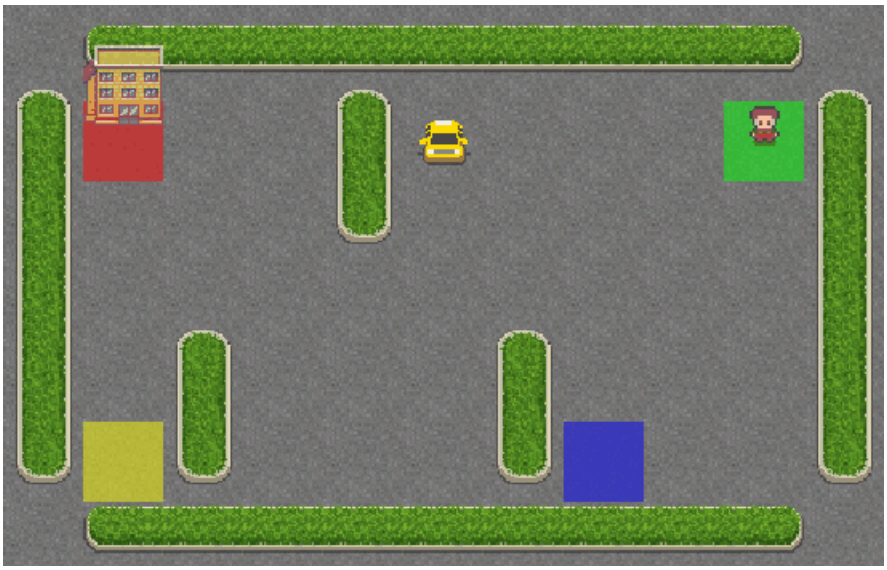


Figure 1: Illustration of the Taxi Domain

The environment for this task is the taxi domain, illustrated in Fig. 1. It is a 5×5 matrix, where each cell is a position your taxi can stay at. There is a single passenger who can be either picked up or dropped off, or is being transported. 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.

1.1 Algorithms

The two algorithms which are used are SMDP Q-Learning and Intra-option Q-Learning:

1.1.1 SMDP Q-Learning

SMDP (Semi-Markov Decision Process) Q-Learning is an extension of Q-learning designed to handle Semi-Markov Decision Processes, where actions can take varying amounts of time to complete. This extension allows the agent to learn policies in environments where actions are not atomic and can persist over multiple time steps.

In an SMDP, the environment is allowed to transition between states at non-homogeneous time intervals. Actions in SMDPs have durations, and transitions between states occur after a certain amount of time.

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(\bar{r}_{t+\tau} + \gamma^{\tau} \max_{a'} Q(s',a') - Q(s,a) \right)$$

$$\bar{r}_{t+\tau} = \sum_{k=0}^{\tau-1} \gamma^{k} r_{t+k+1}$$

1.1.2 Intra-Option Q-Learning

Intra-option Q-Learning addresses situations where actions have varying durations. It incorporates the concept of options, which are temporally extended actions. Options are defined by an initiation set, a termination condition, and a sub-policy. The agent first selects an option, and then follows the sub-policy until the termination condition is met. The formulation for Intra Option Q-Learning is as follows :

$$Q(s,o) = \begin{cases} Q(s,o) + \alpha \left[r_1 + \gamma Q(s',o) - Q(s,o) \right] & \text{if not terminating at } s' \\ Q(s,o) + \alpha \left[r_1 + \max_{a} Q(s',o) - Q(s,o) \right] & \text{if terminating at } s' \end{cases}$$

here the term r_1 refers to the reward obtained upon transition from state s at time t to state s' at time t+1

NOTE :

Taxi-v3
Passenger locations: 0: R(ed); 1: G(reen); 2: Y(ellow); 3: B(lue); 4: in taxi Destinations: 0: R(ed); 1: G(reen); 2: Y(ellow); 3: B(lue) Rewards:

- -1 per step unless other reward is triggered.
- +20 delivering passenger.
- -10 executing "pickup" and "drop-off" actions illegally.

The discount factor is taken to be $\gamma = 0.9$.

2 Results

2.1 SMDP Q-Learning

```
    def SMDP_QLearning(render = None):
    # While episode is not over
    global count_success,eps_main
    state = env.reset()
    rewards = []
    done = False
    total_reward = 0
    T=1
    while not done:
        x, y, passenger, dropoff = env.decode(state)
        sub_state = n_drops * passenger + dropoff

        # Choose option
        option = egreedy_policy(q_values_SMDP, sub_state, epsilon=eps_main)
        eps_main = max(eps_min, eps_decay * eps_main)
        reward_bar = 0
        opt_done = False
        move = 0
        prev_state = state

        x,y,passenger,dropoff = env.decode(state)

        # Execute option
        while not opt_done and not done:
            opt_act, opt_done = Options_frame(env, state, Q_options, option, eps_options[option])

            [x,y,_,_]=  list(env.decode(state))
            next_state, reward, done, _ = env.step(opt_act)
            [x1,y1,_,_]=  list(env.decode(next_state))

            if render is not None:
                clear_output(wait=True)
                print(env.render())
                time.sleep(T)

            reward_bar = gamma * reward_bar + reward
            move += 1
            total_reward += reward


            # Update option epsilon for exploration
            eps_options[option] = max(eps_min, eps_decay * eps_options[option])

            if opt_done:
                reward_surr = 10
            else:
                reward_surr = reward

            if opt_act < 4:
                Q_options[option][5 * x + y, opt_act] += alpha * (reward_surr + gamma * np.max(Q_options[
option][5 * x1 + y1, :]) - Q_options[option][5 * x + y, opt_act])

            state = next_state

        # Update SMDP Q-value
        _, _, passenger, dropoff = env.decode(state)
        sub_state = n_drops * passenger + dropoff

        _, _, passenger, dropoff = env.decode(prev_state)
        sub_prev_state = n_drops * passenger + dropoff

        q_values_SMDP[sub_prev_state, option] += alpha * (reward_bar + (gamma ** move) * np.max(q_values_SMDP
[sub_state, :]) - q_values_SMDP[sub_prev_state, option])
        updates_SMDP[sub_prev_state, option] += 1

    rewards.append(total_reward)

    # Check if passenger dropped off successfully
    x, y, passenger, dropoff = env.decode(state)
```

```
    if passenger == dropoff and render is None:
        count_success += 1
        # clear_output(wait=True)
        print('Success ({}/{}) = {}%'.format(count_success, i + 1, 100 * count_success / (i + 1)))

    return rewards
```

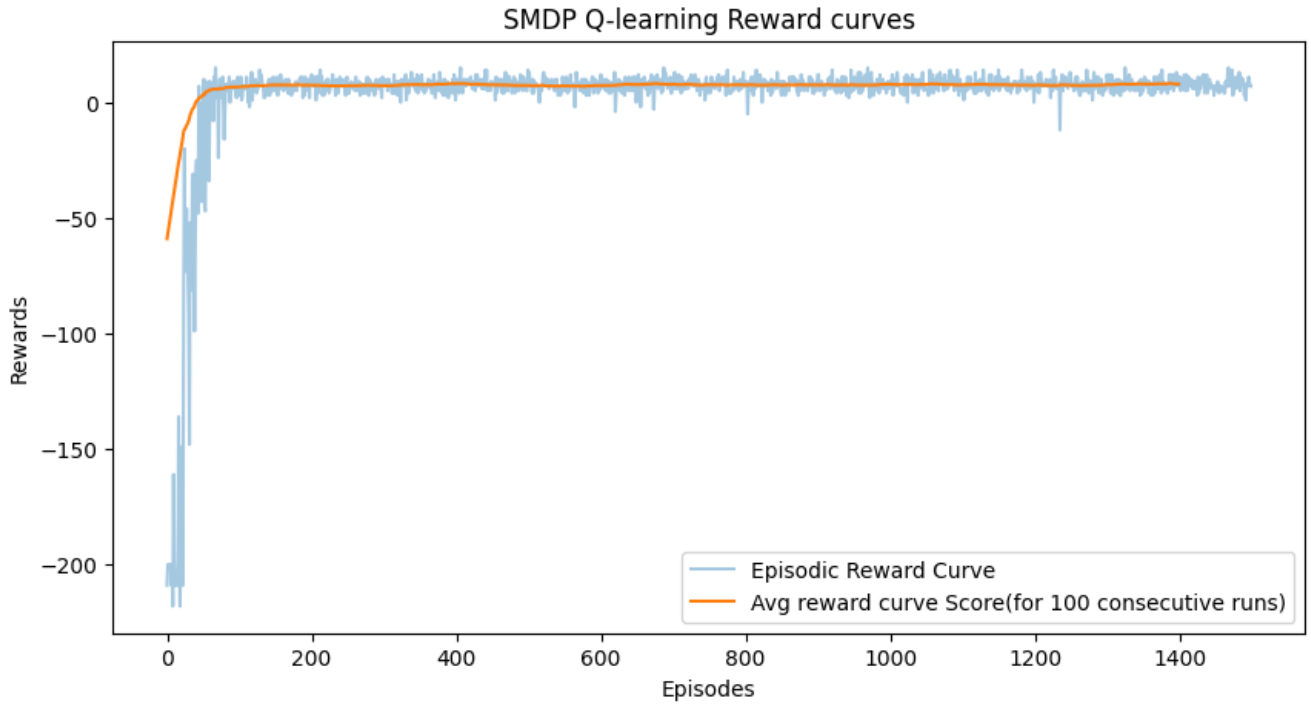


Figure 2: Rewards averaged over 100 runs

Best model
Average timesteps per episode: 17.4773
Average reward for 100 consecutive runs: 7.87
Success Rate: 98.73

From 2, we can see that the average reward for 100 consecutive episodes to converging at **+7.87** which shows :

- **Stability:** The learning process stability indicates that the learning algorithm has reached a relatively consistent performance level over the consecutive runs.
- **Convergence:** The convergence of the average reward implies that the learning algorithm has learned to exploit the environment effectively and has likely converged to a certain policy or behavior that yields the observed average reward.
- **Optimality:** The goal is often to maximize the cumulative reward over time and the average reward has converged to a satisfactory level, which suggests that the learned policy is reasonably effective in achieving the task objectives.

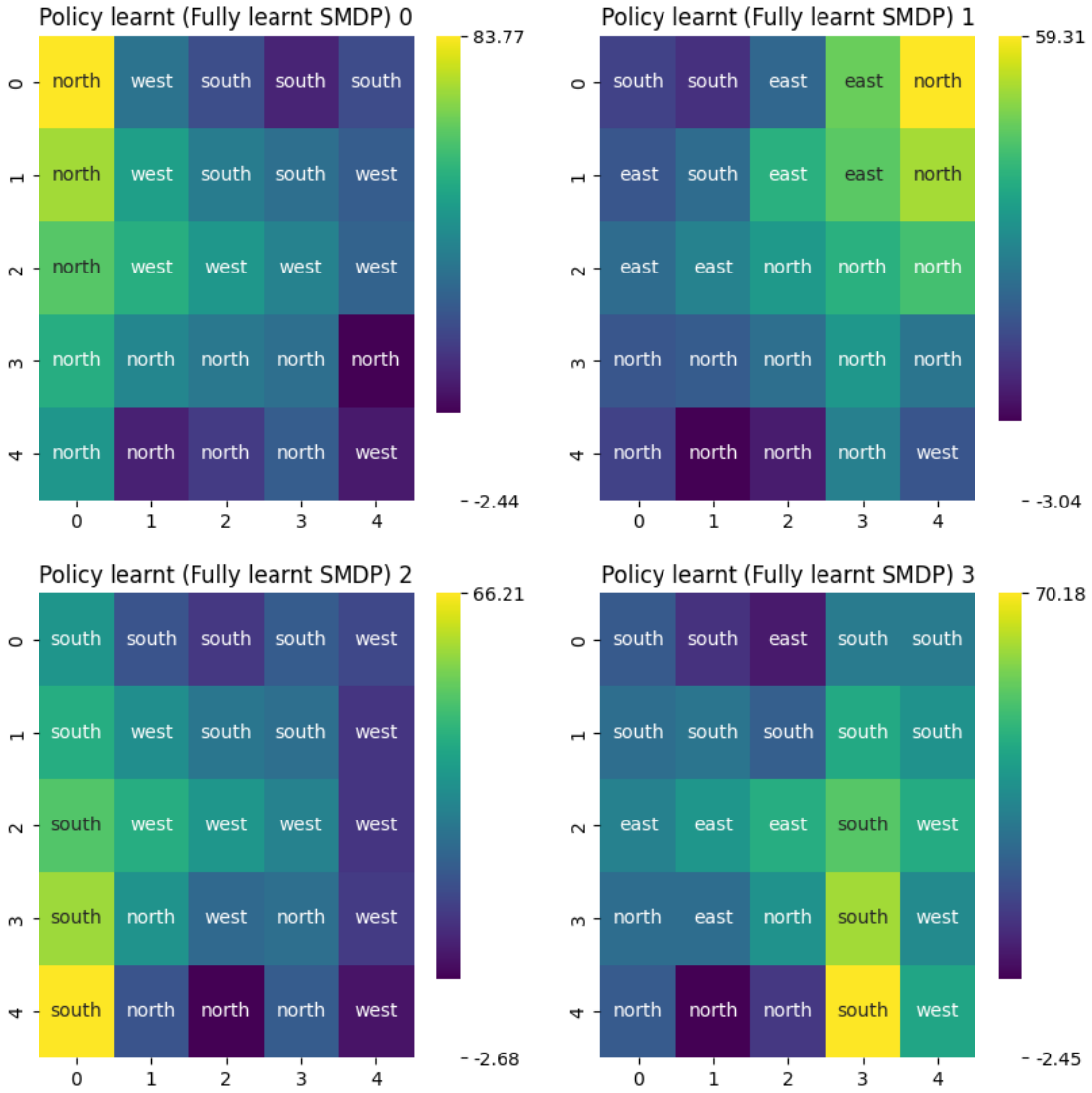


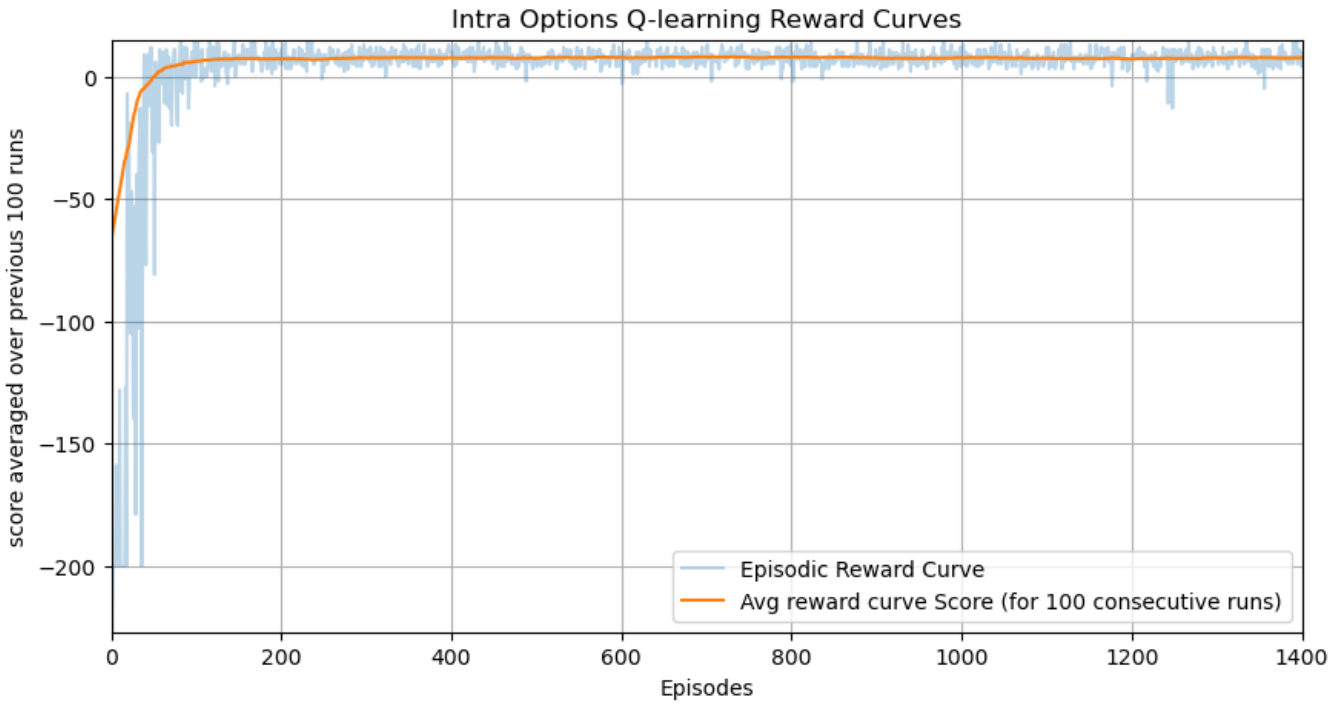
Figure 3: Policy learnt by respective options

- **Options:** 4 options are present to dictate the sequence of actions the taxi should take to get the passenger to that specific location namely, **goto_R**, **goto_G**, **goto_Y**, **goto_B**
- **Policy Description:** Depending on the passenger and drop, an option is chosen using SMDP Q-value and the policy of each options is represented by heatmap in 3.
  - The colored grids represent the taxi's location. The numbers within each grid represent the action the taxi should take according to the policy learned for that specific state.
  - The policy aims to minimize the number of steps taken to reach the destination. So, if the taxi is at specific state, the policy of the corresponding option is such that it reaches the pickup/drop in minimum steps.
  - Eg: In the top left corner (policy for moving the passenger Red), if the taxi is in the bottom-most eastern corner (grid (4,4)), the policy dictates it should move west-north*2-west*3-north*2.
- **Learning Process:** The SMDP Q-learning algorithm learns through trial and error as it explores different actions from each state and updates the Q-value table based on the rewards received and over time, the algorithm learns the most efficient sequence of actions (policy) to take for each option (moving passenger to a specific location) from any state in the environment.

2.2 Intra-Option Q-Learning

```
# Initialize Q-values for each option
Qopt = {i: np.zeros((nX * nY, env.action_space.n - 2)) for i in range(nO)}  # Subtracting pick,drop
q_values_IOQL = np.zeros((nPas* nDrop, nO))
tot_epoch = 0

def IntraOptions_QLearning(render = None):
  global tot_epoch
  rewards = []  # To store rewards per episode
  eps_main = 0.5
  count = 0
  updates_IOQL = np.zeros((nPas* nDrop, nO))
  eps = {i:0.1 for i in range(nO)}
  T = 1

  for i in range(Neps):
    state = env.reset()
    done = False
    tot_rew=0

    while not done:

        _,_,pas,drop = env.decode(state)
        subState = nDrop*pas+drop
        action = egreedy_policy(q_values_IOQL, subState, epsilon=eps_main)
        eps_main = max(eps_min,eps_main*eps_decay)

        option = action
        optdone = False
        prev = state

        while not optdone and not done:

            optact,optdone = Option(env,state,Qopt,option,eps[option])
            next_state, reward, done,_ = env.step(optact)
            tot_rew+=reward
            [x,y,_,_]=  list(env.decode(state))
            [x1,y1,_,_]=  list(env.decode(next_state))

            if render is not None:
                clear_output(wait=True)
                print(env.render())
                time.sleep(T)

            eps[option] = max(eps_min,eps_decay*eps[option])
            if optdone:
              reward_surr = 10
            else :
               reward_surr = reward

            if optact<4:
              Qopt[option][5*x+y, optact] = Qopt[option][5*x+y, optact] + alpha*(reward_surr + gamma*np.max(
    Qopt[option][5*x1+y1, :]) - Qopt[option][5*x+y, optact])

            for o in range(nO):
              optact_o,optdone_o = Option(env,state,Qopt,o,eps[o])
              if optact_o == optact:
                eps[o] = max(eps_min,eps_decay*eps[o])
                if optdone_o:
                  q_values_IOQL[Sub(state), o] += alpha*(reward + gamma*np.max(q_values_IOQL[Sub(next_state),
     :]) - q_values_IOQL[Sub(state), o])
                else:
                  q_values_IOQL[Sub(state), o] += alpha*(reward + gamma*q_values_IOQL[Sub(next_state), o] -
    q_values_IOQL[Sub(state), o])

                updates_IOQL[Sub(state), o] += 1
            tot_epoch += 1
            state = next_state

    rewards.append(tot_rew)
    x,y,pas,drop = env.decode(state)

    if pas==drop:
```

```
        count+=1
        # clear_output(wait=True)
        print('Success ({}/{}) = {}%'.format(count,i+1,100*count/(i+1)))

    return rewards

rewards_ioql = IntraOptions_QLearning()
print(tot_epoch/num_episodes)
```



Best model
Average timesteps per episode: 16.825
Average reward for 100 consecutive runs: 8.12
Success Rate: 98.93

From the above plot we can observe the similar characteristics as that of SMDP Q-Learning :

- Stability
- Convergence
- Optimality

Moreover it converges to a reward of **+8.12** after 1500 episodes, which is higher than that of SMDP Q-learning.

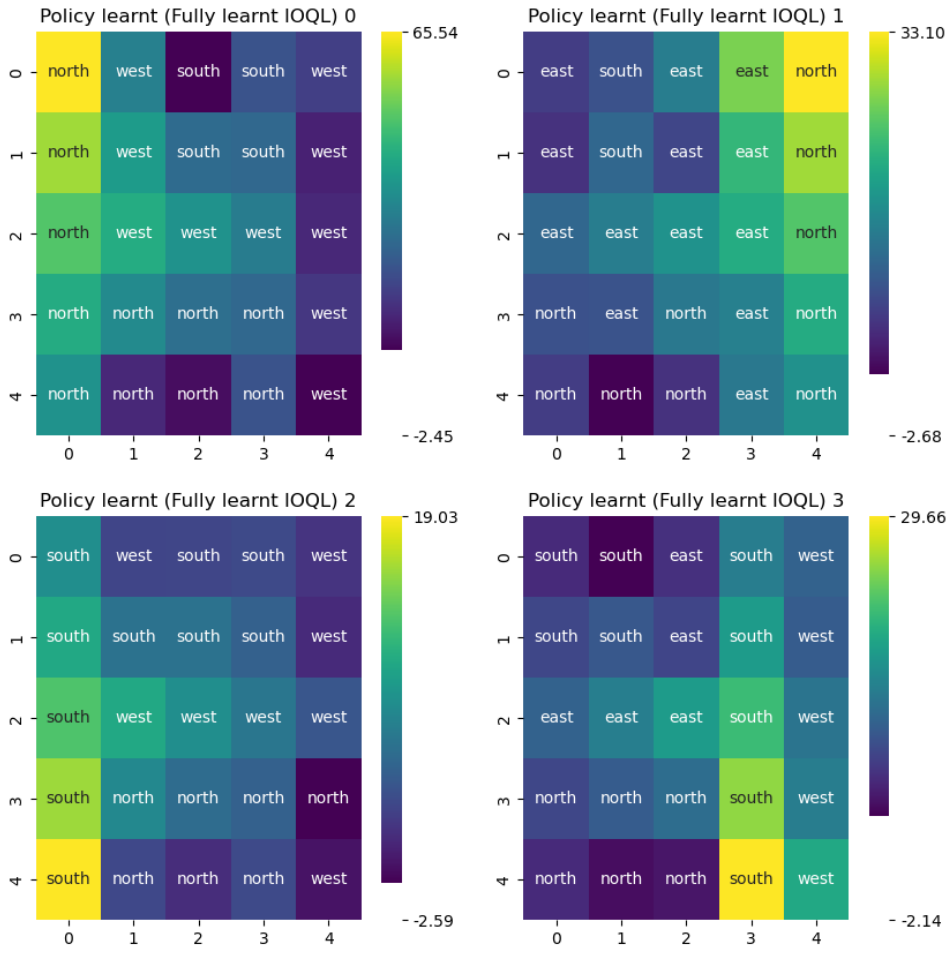


Figure 4: Policy learnt by using IOQL

- **Options:** 4 options are present to dictate the sequence of actions the taxi should take to get the passenger to that specific location namely, **goto_R**, **goto_G**, **goto_Y**, **goto_B**
- **Policy Description:** Depending on the passenger and drop, an option is chosen using Intra Options Q-value and the policy of each options is represented by heatmap in 6.
  - According to this policy, if we follow option 0 [reach (0,0) to terminate option] then for any spawn location of the taxi we can observe the policy dictates it to avoid the obstacles and reach the location [0,0] in way that maximizes the reward obtained.
  - For example, following option 1, the taxi gets spawned at (0,4), then following the learned policy the taxi would perform the following actions : West - 2*South - 3*West - 2*North - option termination, This approach by the taxi to go down first and coming up to reach option termination is due to the obstacles present at (1,1) and (0,1).
  - The same can be observed for other options with different spawn locations as well
- **Learning Process:** The Intra Options Q-learning algorithm learns through trial and error as it explores different actions from each state and updates the Q-value table for every option that takes the selected action at that timestep
- **Reasoning behind Learned Policy:** The policy favors actions that directly contribute to reaching the destination or receiving a positive reward. For example, if the taxi is next to a passenger and the destination is north, the policy might prioritize picking up the passenger (action 4) even if it's slightly out of the way, as this action directly contributes to completing the trip and earning a reward.

2.3 Alternative Options

Given Options:

- `goto_red`
- `goto_green`
- `goto_blue`
- `goto_yellow`

Alternate Options:

- `moveonly_down`
- `moveonly_up`
- `moveonly_left`
- `moveonly_right`

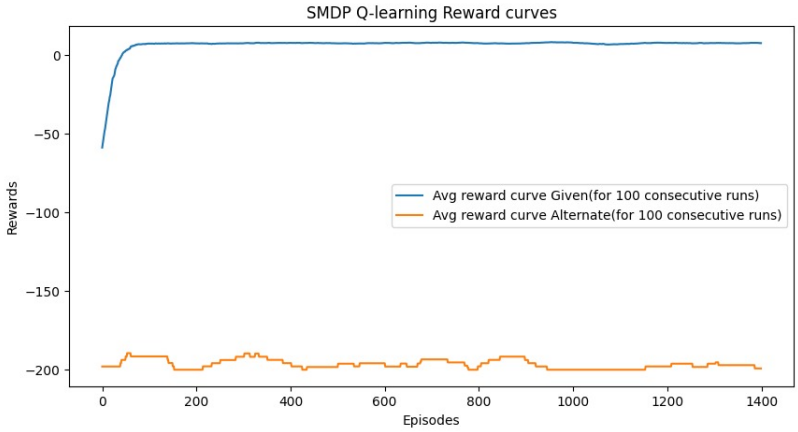


Figure 5: Reward comparison between given and alternate option for SMDP

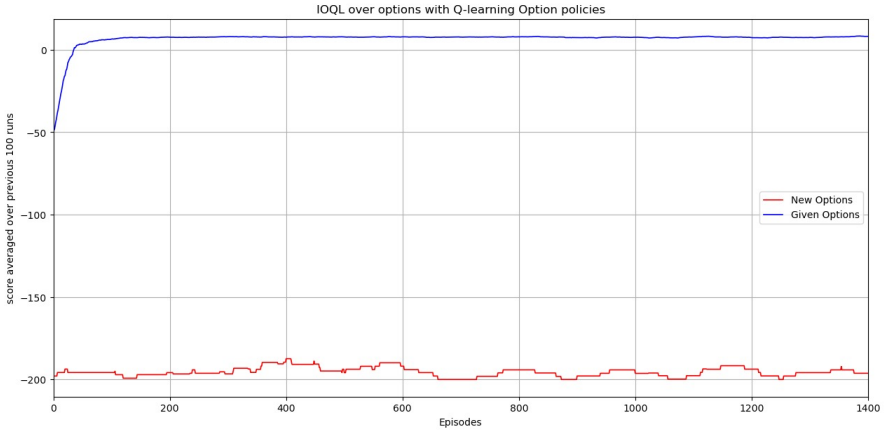


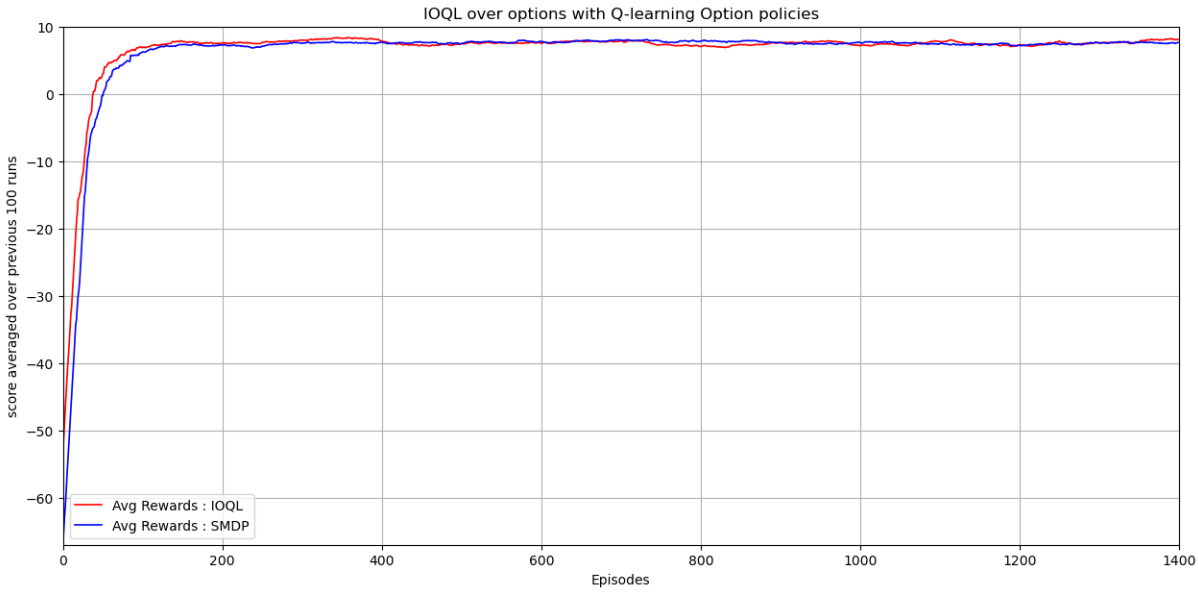
Figure 6: Reward comparison between given and alternate option for IOQL

Performance Comparison:
Reward Curves: The average reward curve for the "Given Option" algorithm (upper curve) is significantly higher than the one for the "Alternate Option" (lower curve). This indicates a substantial difference in performance.

Reasons for Performance Difference:
Structured Approach: The "Given Option" utilizes high-level options that directly target destinations. This structured approach likely helps the agent navigate more efficiently and avoid getting stuck. The options might also implicitly encode some knowledge about the environment's layout, leading to better reward-gathering strategies.

The **"Given Option"** algorithm with designated location options significantly outperforms the **"Alternate Option"** with basic movement actions. This highlights the effectiveness of using higher-level abstractions or options in reinforcement learning tasks.

2.4 Comparison between SMDP and IO Q-Learning



From the comparision plot we can make the following inferences :

- The convergence for Intra Options Q-Learning occurs faster than Semi Markov Decision Process Q-Learning
- This could be due to the way in with the Q table update corresponding to the options are performed, Intra Options would update the option-Q table for every option that follows the same action at any given state whereas SMDP would only update after the option has terminated, still it would only update the option Q value corresponding to that option only