Reinforcement Learning : Programming Assignment 3

Rishabh Vashistha AE23S040 and Bhavik More CH22M009

21st April 2024

Please refer to https://github.com/rishi1906/RL_PA_3.git for codes.

1 Environment(Taxi-v3)

Description

There are four designated pick-up and drop-off locations (Red, Green, Yellow and Blue) in the 5×5 grid world. The taxi starts off at a random square and the passenger at one of the designated locations.

The goal is to move the taxi to the passenger's location, pick up the passenger, move to the passenger's desired destination, and drop off the passenger. Once the passenger is dropped off, the episode ends.

Map:

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

Action Space

The action shape is $(1,)$ in the range $\{0, 5\}$ indicating which direction to move the taxi or to pickup/drop off passengers.

- 0: Move south (down)
- 1: Move north (up)
- 2: Move east (right)
- 3: Move west (left)
- 4: Pickup passenger
- 5: Drop off passenger

Observation Space

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.

Destination on the map are represented with the first letter of the color.

Passenger locations:

- 0: Red
- 1: Green
- 2: Yellow
- 3: Blue
- 4: In taxi

Destinations:

- 0: Red
- 1: Green
- 2: Yellow
- 3: Blue

Note that there are 400 states that can actually be reached during an episode. The missing states correspond to situations in which the passenger is at the same location as their destination, as this typically signals the end of an episode. Four additional states can be observed right after a successful episodes, when both the passenger and the taxi are at the destination. This gives a total of 404 reachable discrete states.

Starting State

The episode starts with the player in a random state.

Rewards

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

Episode End

The episode ends if:

- The taxi drops off the passenger.

2 Tasks

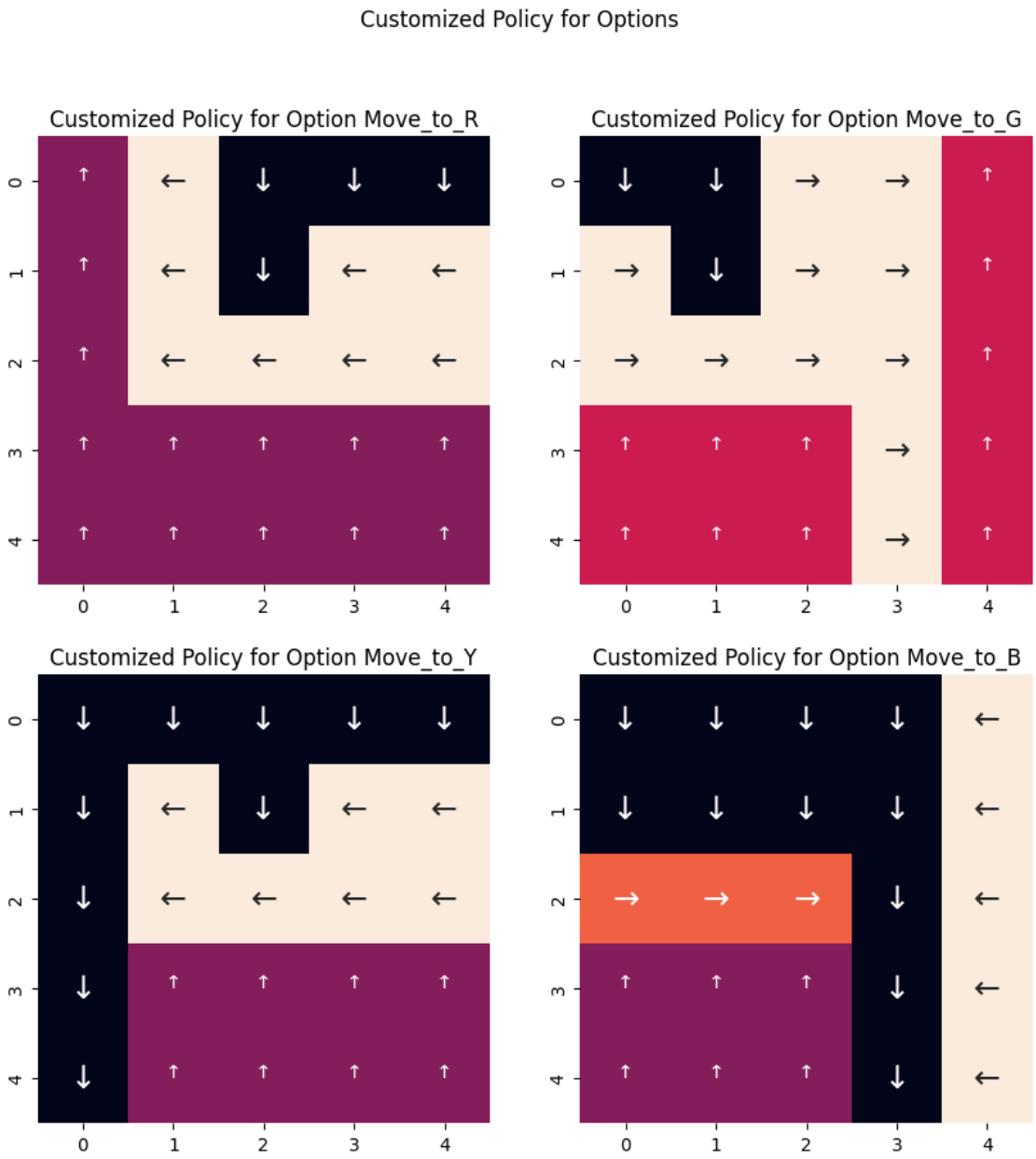


Figure 1: Policy for Option Set 1

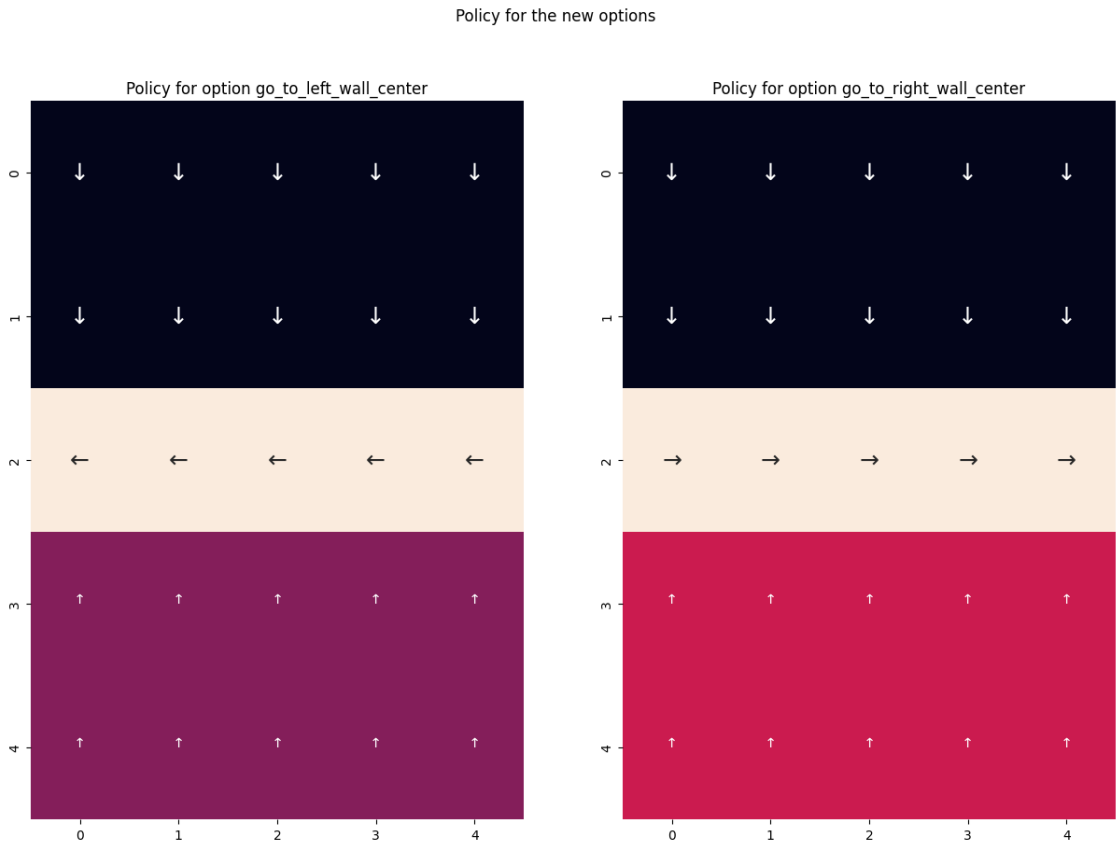


Figure 2: Policy for Option Set 2

- Manually Crafted Options
  - The set of four plots displayed on the top illustrates manually crafted options.
  - The problem description offers choices for moving the taxi to each of the four specified locations: R, G, Y, B, which can be executed when the taxi is not already at one of these locations.
  - **OPTION SET 1**: `move_to_R`, `move_to_G`, `move_to_Y`, `move_to_B` demonstrate the basic action required for the taxi to move from any position to reach R, G, B and Y respectively, within the 5×5 grid. Visual representations of the policies for the OPTIONS SET 1 are given in Figure 1.
  - **OPTION SET 2**:(Alternate options)The newly introduced options direct the taxi to either the left center(M) or the right center of the grid(N). This approach is promising as locations R and Y are equidistant to the left center, while locations G and B are equidistant to the right center. Visual representations of the policies for the options `move_to_M` and `move_to_N` are provided in Figure 2.

```
+---------+
|R: | : :G|
| : | : : |
|M: : : : |
| | : | : |
|Y| : |B: |
+---------+
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : :N|
| | : | : |
|Y| : |B: |
+---------+
```

The Reward curves and Visualizations of the learned Q-values(Task1), and Written description of the policies learnt and Our reasoning behind why the respective algorithm learns the policy are covered in SECTION 3 and 4 for the option sets 1 and 2 respectively.

3 Hyper Parameter Tuning

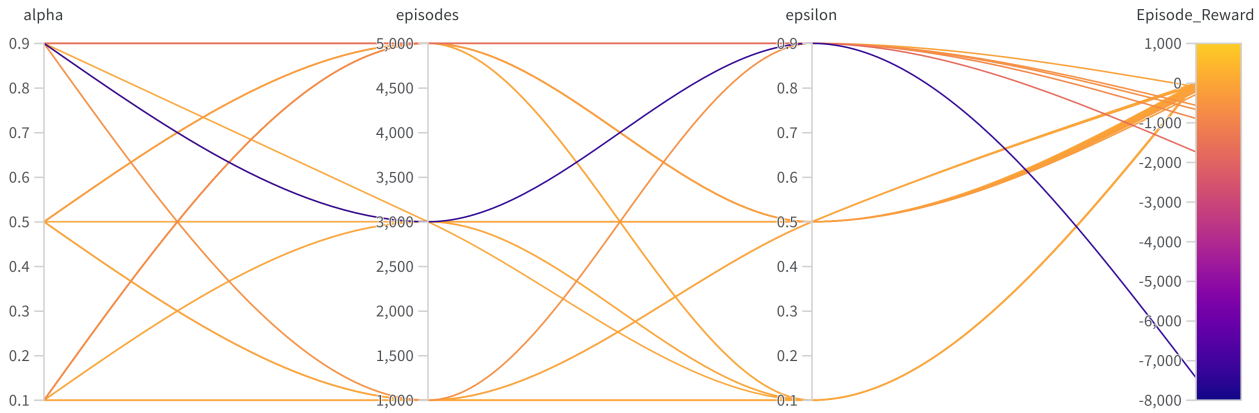


Figure 3: Hyper-parameter Tuning using WANDB for SMDP Q-Learning

The above sweep diagrams are for the WandB hyperparameter sweeps for optimal Episode reward for SMDP Q-Learning The range of values for which we tuned the hyperparameters for SMDP Q-Learning are:

- Exploration factor ϵ: [0.1, 0.5, 0.9]
- Learning rate α: [0.1, 0.5, 0.9]
- Number of Episodes: [1000, 3000, 5000]

The optimal hyperparameters which yielded best Episode reward amongst the combinations of the hyperparameters for SMDP Q-Learning are:

- Exploration factor ϵ: [0.1]
- Learning rate α: [0.5]
- Number of Episodes: [5000]

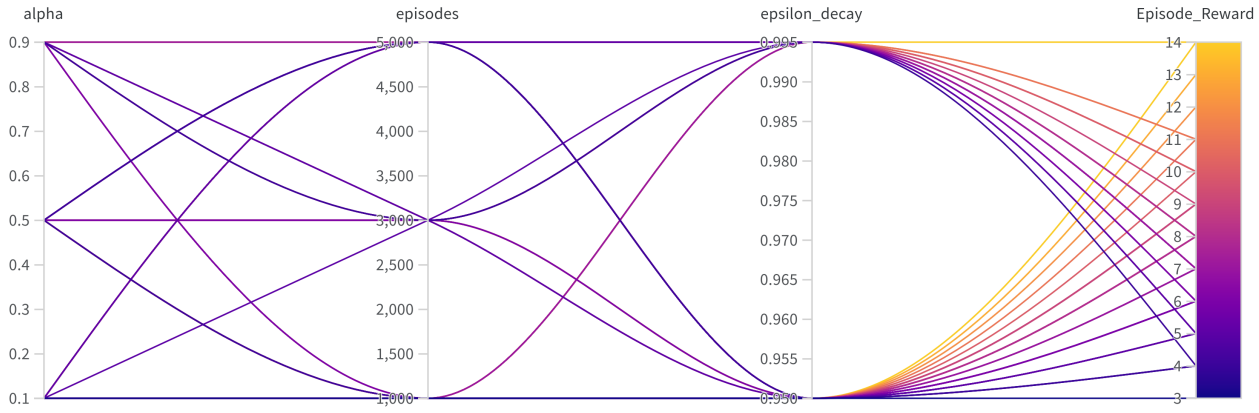


Figure 4: Hyper-parameter Tuning using WANDB for IOQL Q-Learning

The above sweep diagrams are for the WandB hyper-parameter sweeps for optimal Episode reward for Intra-Option Q-Learning.

The range of values for which we tuned the hyper-parameters are:

- Exploration factor ϵ-decay: [0.995, 0.95]
- Learning rate α: [0.1, 0.5, 0.9]
- Number of Episodes: [1000, 3000, 5000]

The optimal values which yielded best Episode reward amongst the combinations of the hyper parameters for Intra-Option Q-Learning are:

- Exploration factor ϵ-decay: [0.995]
- Learning rate α: [0.9]
- Number of Episodes: [5000]

4 Code Snippets

4.1 Implementation details of SMDP Single Step Q Learning

```
# Define a function to update Q-values for primitive actions
update_q_primitive = lambda stt, act, nxt_stt, reward: \
    SMDP_QV[stt][act] + alpha * (reward + gamma * np.max(SMDP_QV[nxt_stt]) - SMDP_QV[stt][act])

update_q_option = lambda int_stt, act, tau, nxt_stt, reward_bar: \
    SMDP_QV[int_stt][act] + alpha * \
    (reward_bar + (gamma ** tau) * np.max(SMDP_QV[nxt_stt]) - SMDP_QV[int_stt][act])
```

Listing 1: Python code for updating Q-values

```
# Define a function to perform SMDP Q-Learning for options
def smdp_q_learning_option(act_func, stt, done):
    optdone = False
    int_stt = stt
    tau = 0
    reward_bar = 0
    while (optdone == False and not done):
        optact, optdone = act_func(env, stt)
        # print(optact)
        nxt_stt, reward, done, _, _ = env.step(optact)
        reward_bar = gamma * reward_bar + reward
        tau += 1
        stt = nxt_stt
    return int_stt, tau, reward_bar, nxt_stt
```

Listing 2: Python code for SMDP Q-Learning for options

```
def run_smdp_q_learning(actions_funcs, no_act, alpha, eps, exp, n_eps):
    cumulative_reward = 0
    for i in range(exp):
        for episode in tqdm(range(n_eps)):
            step = 0
            stt = env.reset()[0]
            done = False
            total_rewards = 0

            while not done:
                step += 1
                act = epsilon_greedy(SMDP_QV, stt, eps, no_act)

                if act < 6:
                    nxt_stt, reward, done, _, _ = env.step(act)
                    total_rewards += reward
                    SMDP_QV[stt][act] = update_q_primitive(stt, act, nxt_stt, reward)
                    stt = nxt_stt

                else:
                    act_func = actions_funcs[act - 6]
                    int_stt, tau, reward_bar, nxt_stt = smdp_q_learning_option(act_func, stt, done)
                    SMDP_QV[int_stt][act] = update_q_option(int_stt, act, tau, nxt_stt, reward_bar)
                    total_rewards += reward_bar

            cumulative_reward += total_rewards
            rewards[i][episode] = total_rewards
            steps.append(step)
            r = np.sum(rewards, 0)
            mean_reward = np.mean(r)
            wb.log({'Episode_Reward': total_rewards})
            wb.log({'Episode_Step': step})
            wb.log({'Mean_Episodic_Reward': mean_reward})
```

Listing 3: Python code for running SMDP Q-Learning

```
sweep_config={
    'method': 'bayes',
    'metric': {'name': 'Mean_Episodic_Reward', 'goal': 'maximize'},
    'parameters': {
        'alpha': {'values': [0.1, 0.5, 0.9]},
        'epsilon': {'values': [0.1, 0.5, 0.9]},
        'episodes': {'values': [1000, 3000, 5000]},
    }
}
project_name = 'PA3_SMDP'
sweep_id = wb.sweep(sweep_config, project=project_name)
```

Listing 4: Python code for hyperparameter sweep configuration

```
def train():
  with wb.init() as run:
    config = wb.config
    actions = [move_to_R, move_to_G, move_to_Y, move_to_B]
    run_smdp_q_learning(actions, no_act, config.alpha, config.epsilon, exp, config.episodes)

wb.agent(sweep_id, function=train, count=50)
wb.finish()
```

Listing 5: Python code for training with Weights and Biases

4.2 Implementation details of IOQL Single Step Q Learning

```
update_option = lambda nxt_stt, optdone, act: ((optdone) * np.max(IOQL_QV[nxt_stt]) + (1 - optdone) * (
    IOQL_QV[nxt_stt][act]))
```

Listing 6: Python code for the lambda function IOQL update option

```
from tqdm import tqdm

def run_IOQL(env, IOQL_QV, action_funcs, eps_min, eps_, eps_decay, alpha, gamma, exp, n_eps, no_act):
    cumulative_reward = 0
    for i in range(exp):
        for episode in tqdm(range(n_eps)):
            step = 0
            stt = env.reset()[0]
            done = False
            total_rewards = 0

            while not done:
                step += 1
                eps = min(eps_min, eps_)
                act = epsilon_greedy(IOQL_QV, stt, eps, no_act)
                eps_ *= eps_decay

                if act < 6:
                    nxt_stt, reward, done, _, _ = env.step(act)
                    total_rewards += reward
                    IOQL_QV[stt][act] += alpha * (reward + \
                    gamma * np.max(IOQL_QV[nxt_stt]) - IOQL_QV[stt][act])
                    stt = nxt_stt
                else:
                    act_func = action_funcs[act - 6]
                    optdone = False
                    int_stt = stt
                    tau = 0
                    reward_bar = 0

                    while not (optdone or done):
                        optact, optdone = act_func(env, stt)
                        next_stt, reward, done, _, _ = env.step(optact)

                        IOQL_QV[stt][optact] += alpha * \
                        (reward + gamma * np.max(IOQL_QV[next_stt]) - IOQL_QV[stt][optact])
                        IOQL_QV[stt][act] += alpha * \
                        (reward + gamma * update_option(next_stt, optdone, act) - IOQL_QV[stt][act])
                        taxi_row, taxi_col, passenger_index, destination_index = env.decode(stt)
                        sim_opt = act + 1 if (act % 2) == 0 else act - 1

                        if taxi_col > 0 and (act==6 or act==7):
                            IOQL_QV[stt][sim_opt] += alpha * \
                            (reward + gamma * np.max(IOQL_QV[next_stt]) - IOQL_QV[stt][sim_opt])
                        elif taxi_col < 3 and (act==8 or act==9):
                            IOQL_QV[stt][sim_opt] += alpha * \
                            (reward + gamma * np.max(IOQL_QV[next_stt]) - IOQL_QV[stt][sim_opt])
                        reward_bar = gamma * reward_bar + reward
                        stt = next_stt

                    total_rewards += reward_bar

            cumulative_reward += total_rewards
            IOQL_rewards[i][episode] = total_rewards
            steps.append(step)
            r = np.sum(rewards, 0)
            mean_reward = np.mean(r)
            wb.log({'Episode_Reward': total_rewards})
            wb.log({'Episode_Step': step})
            wb.log({'Mean_Episodic_Reward': mean_reward})
```

Listing 7: Python code for the IOQL algorithm

```
sweep_config={
    'method': 'bayes',
    'metric': {'name': 'Mean_Episodic_Reward', 'goal': 'maximize'},
    'parameters': {
        'alpha': {'values': [0.1, 0.5, 0.9]},
        'epsilon_decay': {'values': [0.995, 0.95]},
        'episodes': {'values': [1000, 3000, 5000]},
    }
}
project_name = "PA3_IOQL"
sweep_id = wb.sweep(sweep_config, project=project_name)
```

Listing 8: Python code for hyperparameter sweep configuration for IOQL

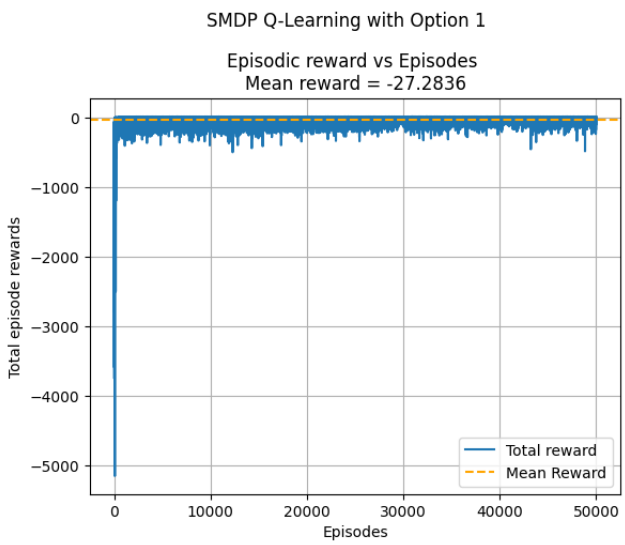
```
def train():
  with wb.init() as run:
    config = wb.config
    action_funcs = [move_to_R, move_to_G, move_to_Y, move_to_B]
    run_IOQL(env, IOQL_QV, action_funcs, eps_min, eps_, config.epsilon_decay, config.alpha, gamma, exp,
     config.episodes, no_act)

wb.agent(sweep_id, function=train, count=50)
wb.finish()
```

Listing 9: Python code for training with hyperparameter sweep for IOQL

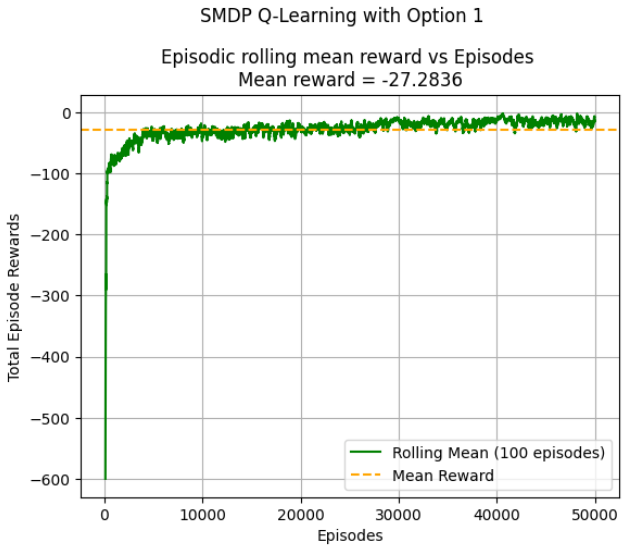
5 SMDP Q-Learning and Intra-Option Qlearning for OPTION SET 1

5.1 SMDP Q-Learning with Option Set 1

5.1.1 Reward curves and visualization of learned Q-values

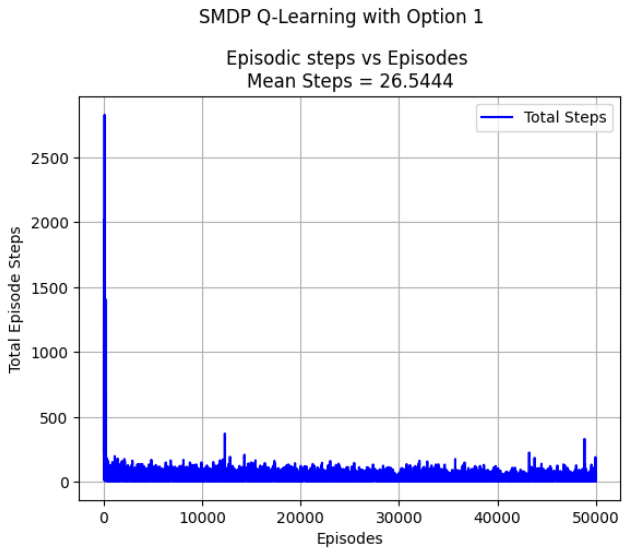


(a) Episodic Reward vs Episodes

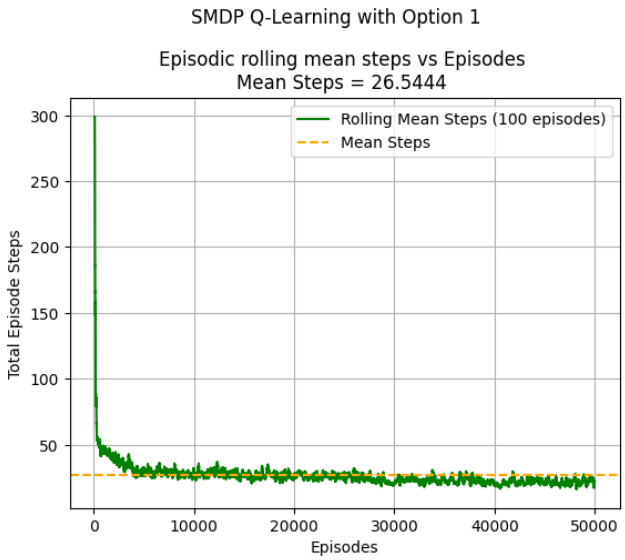


(b) Rolling mean Episodic Reward vs Episodes

Figure 5: Reward Plot for SMDP Q-Learning with Option Set 1



(a) Episodic Steps vs Episodes



(b) Rolling mean Episodic Steps vs Episodes

Figure 6: Steps Plot for SMDP Q-Learning with Option Set 1

- General inferences of SMDP Q-Learning
  - SMDP Q-Learning performs better than normal Q Learning by enabling the use of options: high-level actions that can span multiple time steps.
  - By using options, the agent reduces the number of primitive actions it needs to take to reach a goal. This reduces the number of interactions with the environment required for learning, making it more sample-efficient.
  - Options in SMDP Q-Learning act as a form of "chunking" for exploration. Instead of blindly trying every possible action one by one, the agent can explore larger portions of the state space by executing options. This leads to a more focused and efficient exploration, accelerating learning.
  - The main benifits that we observed as compared to Q-Learning are faster convergence, lower sample complexity (less data needed) and more focused exploration

Figure 7: Learned Q-Values for SMDP Q-Learning with Option Set 1 for Passenger Location : R

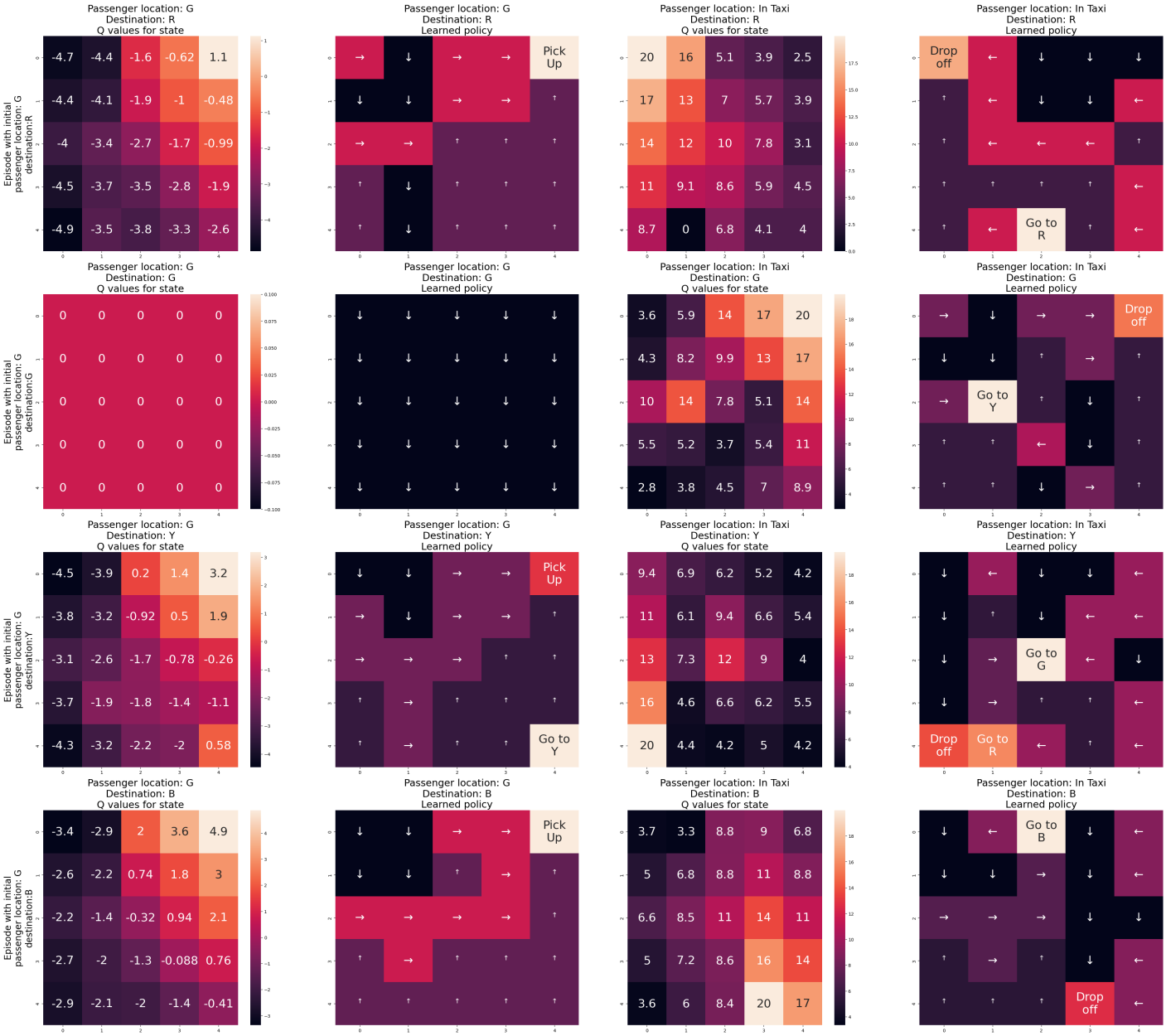


Figure 8: Learned Q-Values for SMDP Q-Learning with Option Set 1 for Passenger Location : G

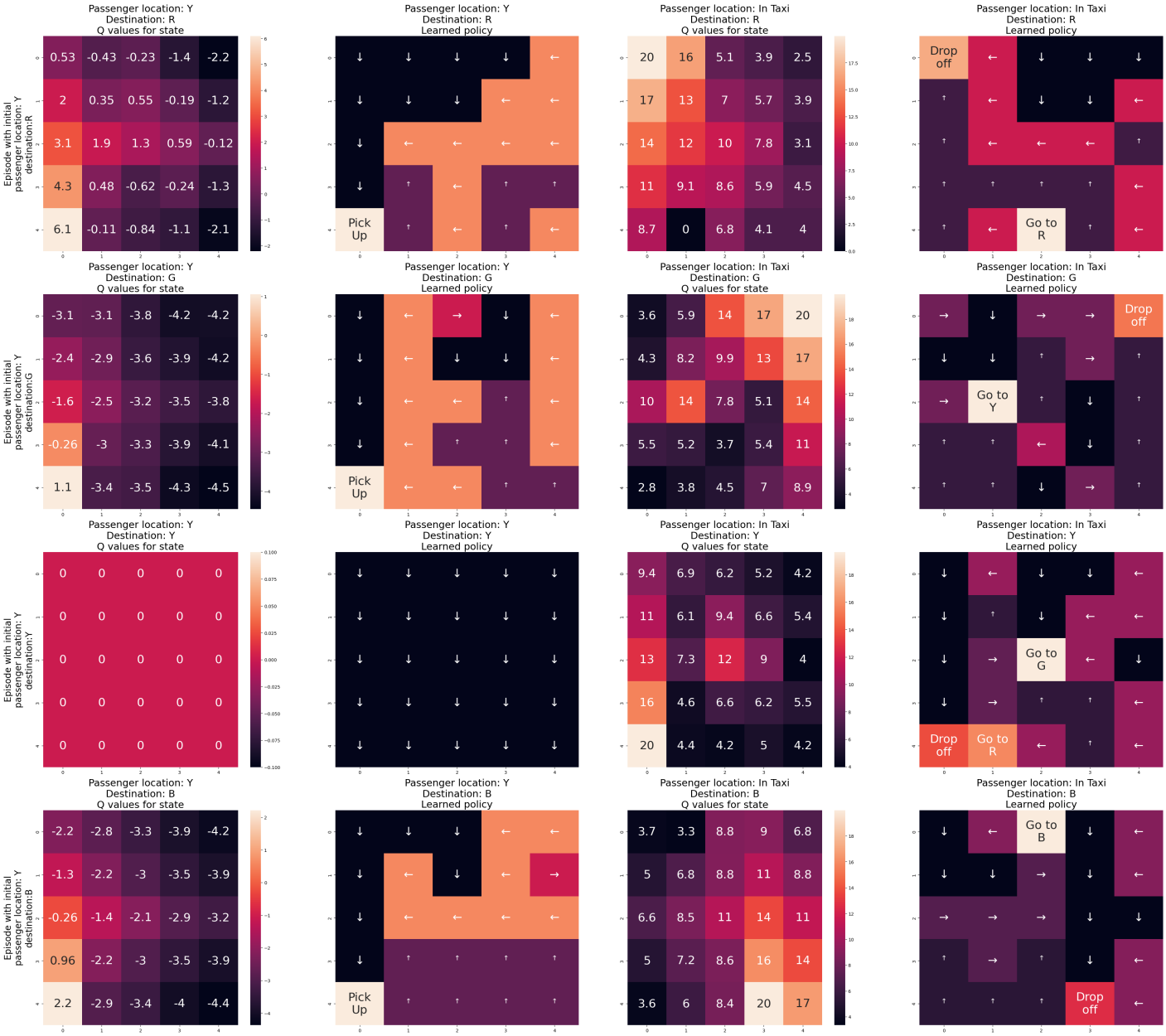


Figure 9: Learned Q-Values for SMDP Q-Learning with Option Set 1 for Passenger Location : Y

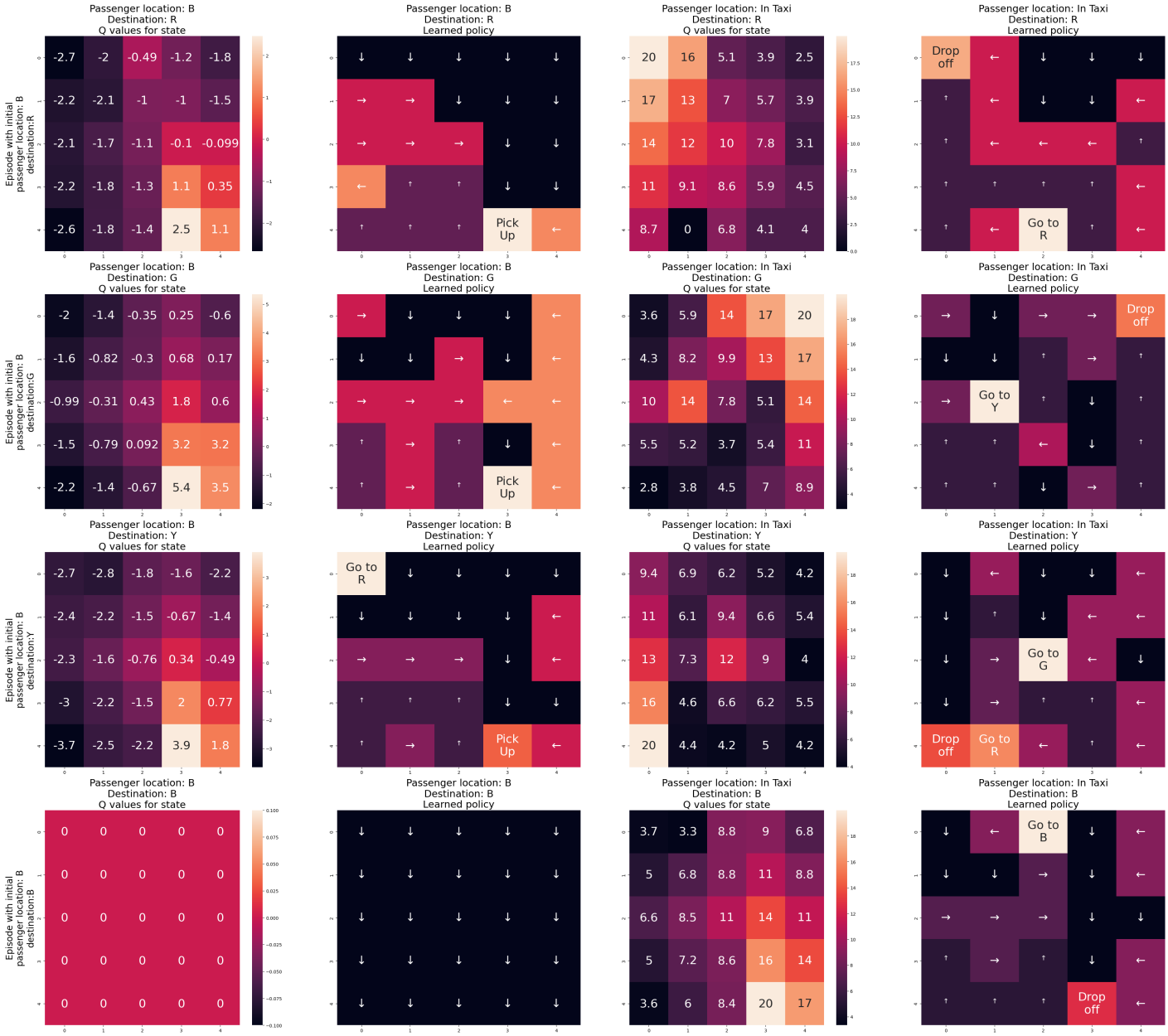


Figure 10: Learned Q-Values for SMDP Q-Learning with Option Set 1 for Passenger Location : B

5.1.2 Inferences from learnt policies

Description of learnt policy:

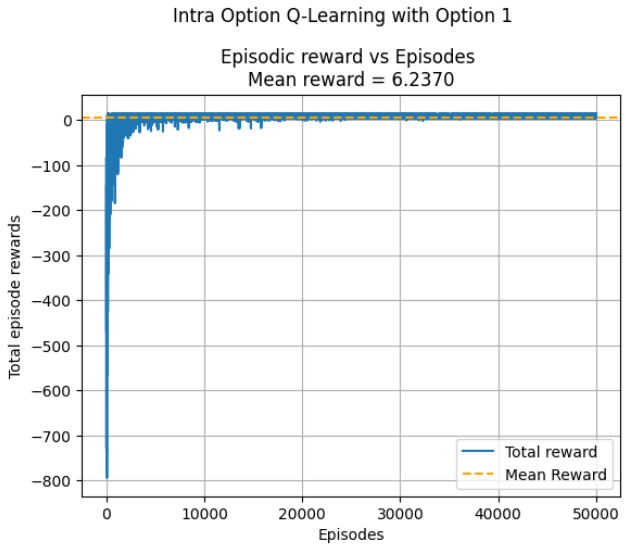
- In above 4 figures, the Q values for states and the corresponding learned policy are visualized.
- Each row corresponds to an initial episode configuration, with fixed passenger and destination locations as specified by the y-label of the first plot of that row.
- Each episode can be broken down into two parts:
  1. The taxi goes to the passenger location and picks up the passenger.
  2. Now, with the passenger in the taxi, the taxi moves to the destination and drops the passenger.
- In each row, the first and second plot correspond to the Q values for state and the learned policy, respectively, for the case when the taxi goes to the passenger location to pick up the passenger. The third and fourth plot correspond to the Q values for state and the learned policy, respectively, for the case where the passenger is in the taxi and the taxi moves to the destination to drop the passenger.
- Some state value graphs would have all their values to be zero. This is because the passenger location and the destination are the same, and the episode terminates immediately as the passenger is already at the destination. These are the $5 \times 5 \times 4 = 100$ unreachable states in an episode. Hence, there are only 400 reachable states that can be explored.
- The agent engages in exploration of the environment prior to picking up a passenger, but reduces its exploration significantly afterward. This shift in behavior is influenced by the high reward associated with dropping off a passenger (+20), which diminishes the necessity for continued exploration.
- Conversely, before picking up a passenger, the agent must thoroughly explore the environment to locate the passenger, considering the penalties incurred for each step (-1) and for picking up the passenger (-1). Hence, the agent exhibits a higher inclination towards exploration before picking up a passenger compared to after.
- The agent swiftly learns to avoid picking up or dropping off passengers at incorrect locations due to the significant negative reward associated with these actions (-10).
- In some cases, the agent chooses an option over primitive action for picking up passenger for instance in Figure 7 second row- Here the agent chooses" Go to R" option to pick passenger from location R even from far of steps.

Reasoning: Why SMDP Q-Learning learns the policy:

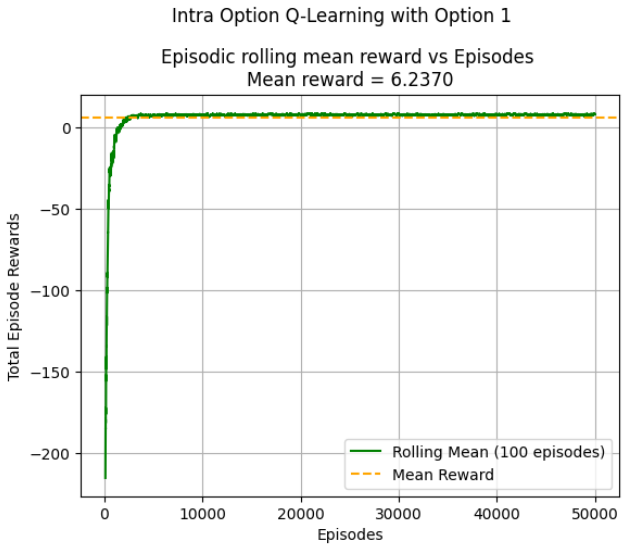
- For SMDP Q-Learning we got optimal results from using $\epsilon = 0.1$. We can deduce that it effectively learns a nearly optimal policy for picking up passengers, but tends to adopt a suboptimal policy for dropping off passengers after pickup, primarily due to reduced exploration post-pickup.
- This limitation suggests that the agent could benefit from ongoing exploration beyond the pickup phase. One proposed solution involves adjusting the reward structure, such as by introducing a smaller negative reward per step, for instance, -0.1 per step.
- In cases when the agent had to drop the passenger at location Y(and G) from the center(and nearby cells) the agent follows the option which takes it to a diagonally opposite location.This behaviour could be due to less exploration(0.1) of the search space.But while tuning the hyper parameters we didn't observed that this behaviour for larger values of epsilon.

5.2 Intra-Option Q-Learning with Option Set 1

5.2.1 Reward curves and Visualization of learned Q-values

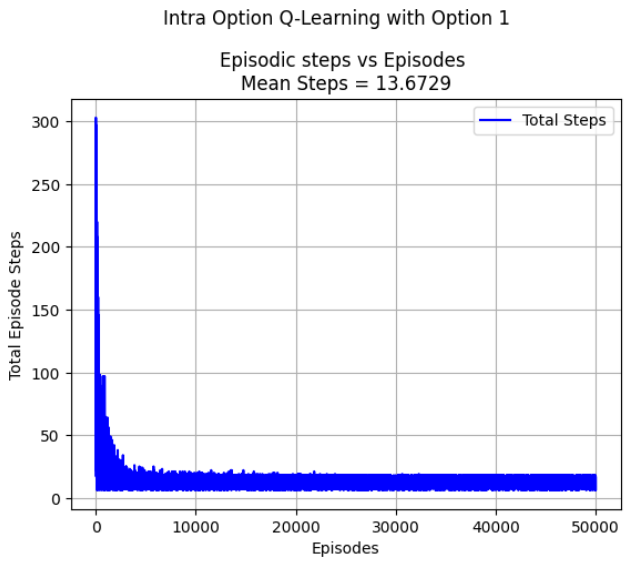


(a) Episodic Reward vs Episodes

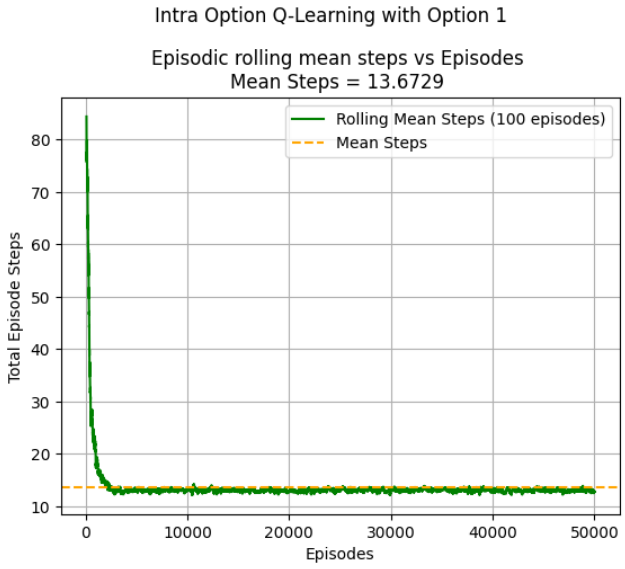


(b) Rolling mean Episodic Reward vs Episodes

Figure 11: Reward Plot for IOQL Q-Learning with Option Set 1



(a) Episodic Steps vs Episodes



(b) Rolling mean Episodic Steps vs Episodes

Figure 12: Steps Plot for IOQL Q-Learning with Option Set 1

- General inferences of Intra Option Q-Learning
  - Intra-option Q-learning converges faster than traditional Q-learning but slower than the SMDP method .
  - This conclusion was drawn from experiments with varying numbers of episodes for both algorithms.
  - Intra-option Q-learning updates state-action values while executing options, requiring multiple updates in addition to learning the options themselves.
  - Compared to traditional Q-learning, intra-option Q-learning with options demonstrates greater sample efficiency.

– This is because it allows the agent to execute higher-level options spanning multiple time-steps, thereby reducing the need for primitive actions.

– Consequently, the learning process accelerates and the number of interactions with the environment needed to learn an optimal policy decreases.

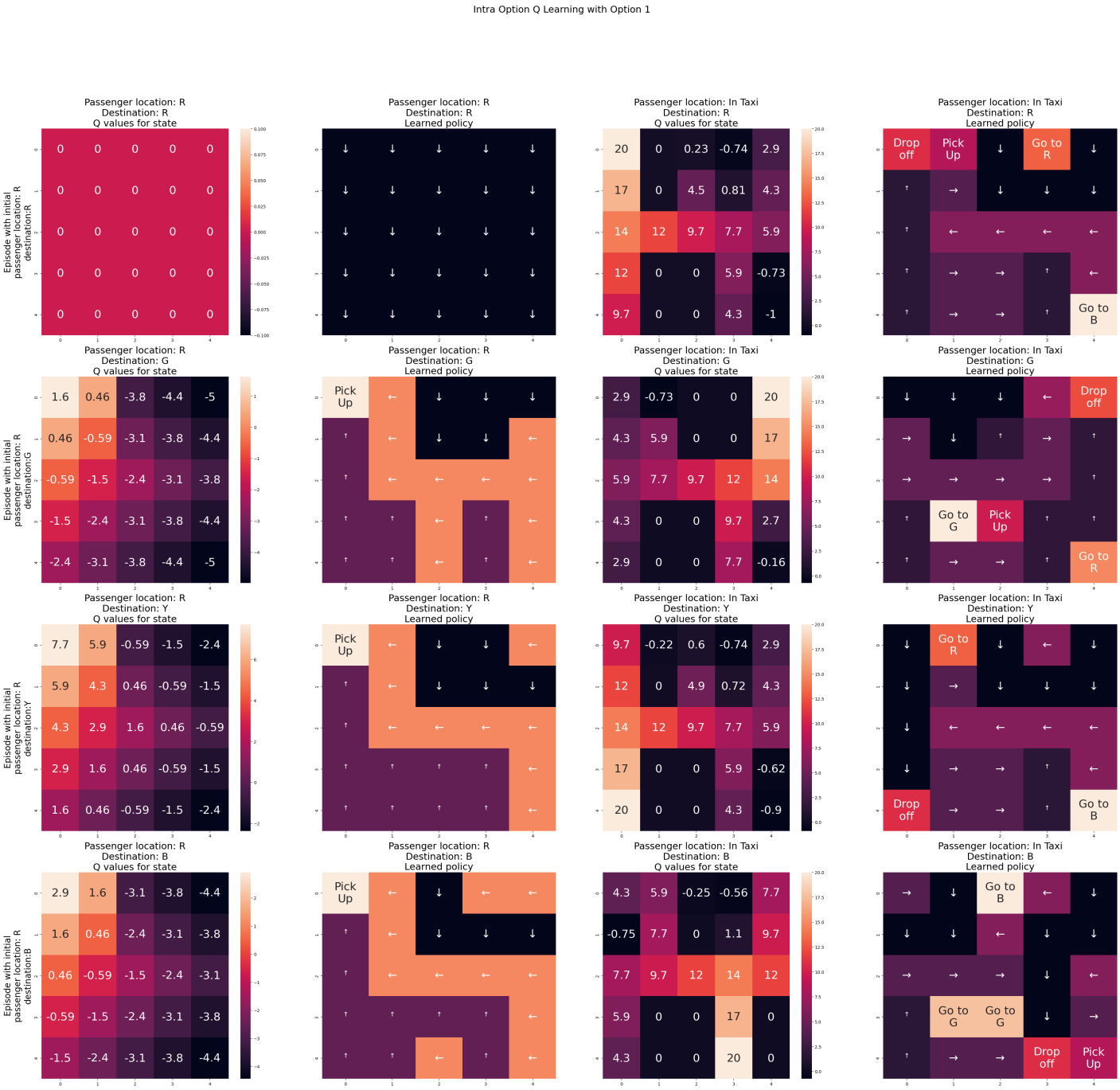


Figure 13: Learned Q-Values for IOQL Q-Learning with Option Set 1 for Passenger Location : R

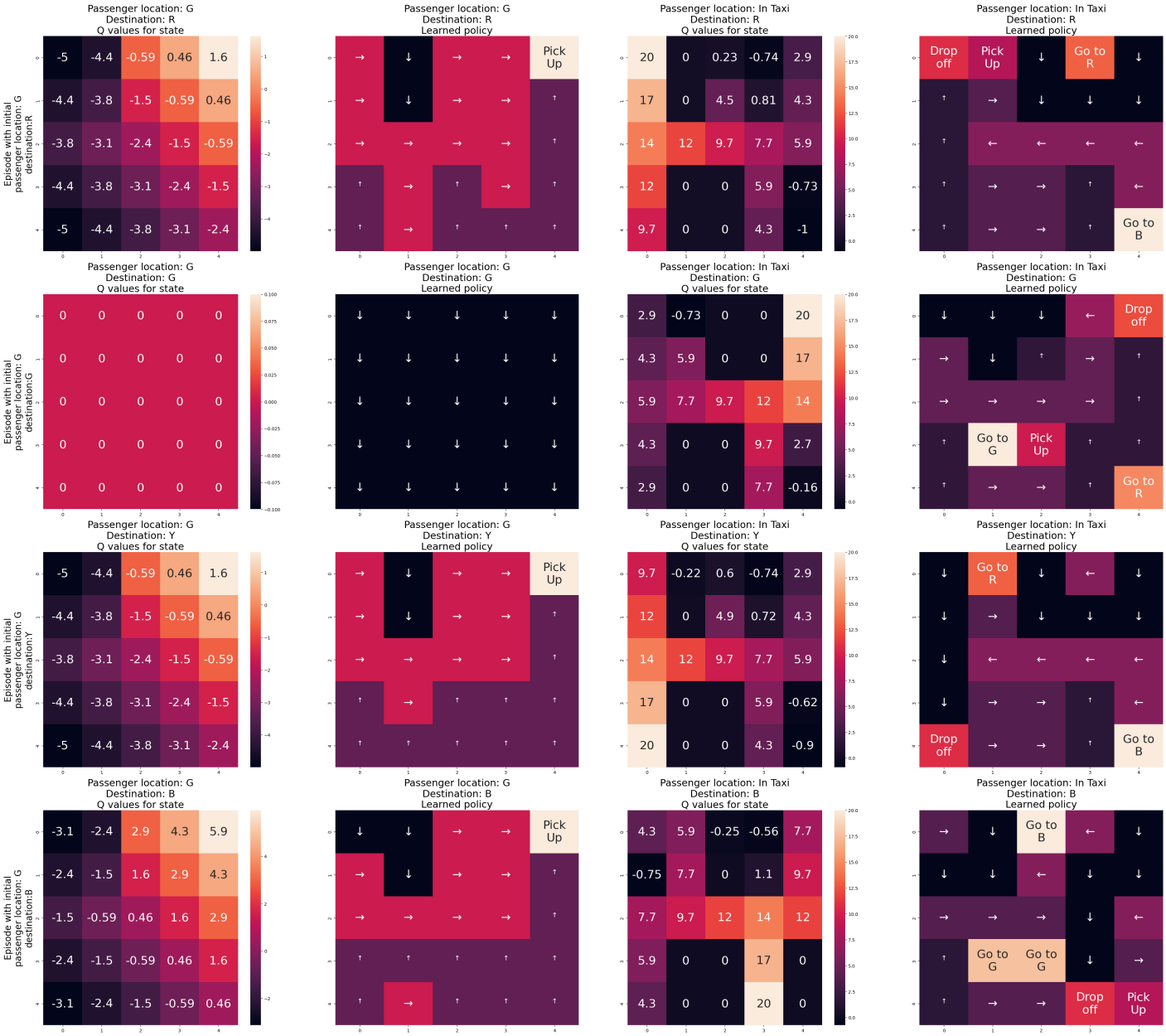


Figure 14: Learned Q-Values for IOQL Q-Learning with Option Set 1 for Passenger Location : G

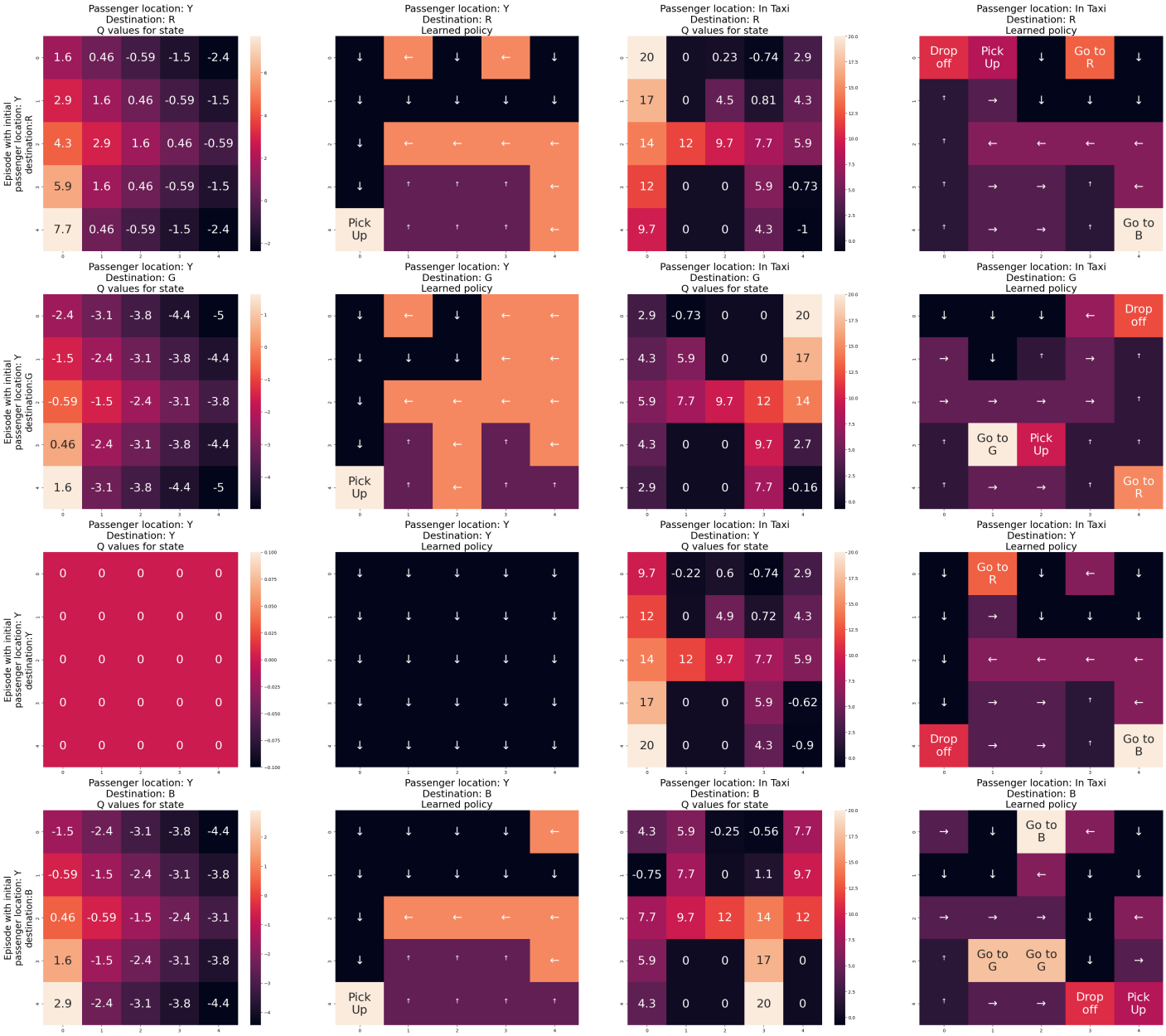


Figure 15: Learned Q-Values for IOQL Q-Learning with Option Set 1 for Passenger Location : Y

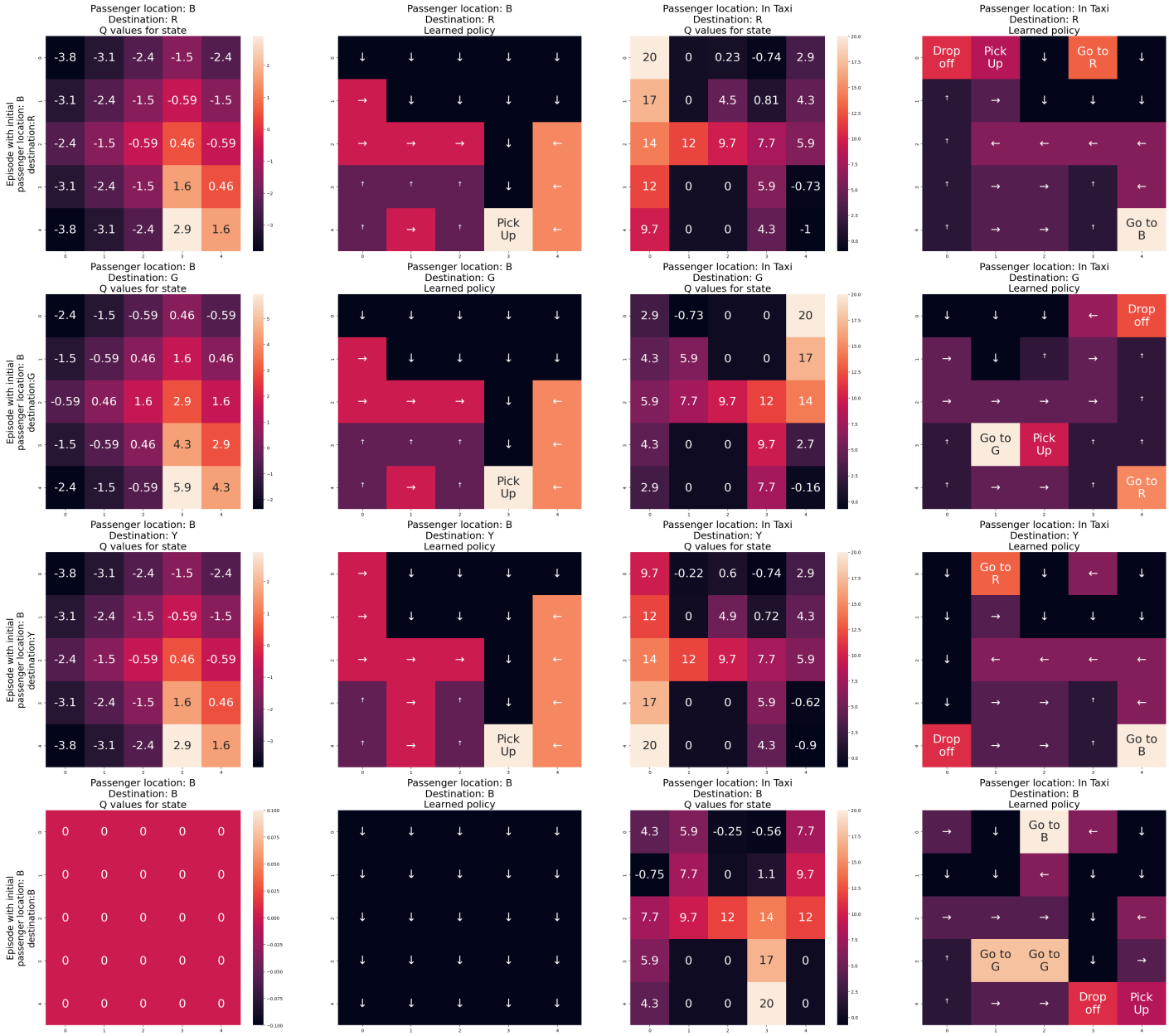


Figure 16: Learned Q-Values for IOQL Q-Learning with Option Set 1 for Passenger Location : B

5.2.2 Inferences from Learnt Policies

Description of Learnt Policy:

- The agent engages in more extensive exploration of the environment before picking up the passenger, but this exploration diminishes after the passenger is picked up.
- Before picking up a passenger, the agent must thoroughly explore the environment to locate the passenger.
- Due to the large negative reward (-10) associated with picking up or dropping off passengers at wrong locations, the agent quickly learns to avoid these actions.
- While certain cells in the plots may indicate "pick up passenger" or "drop off passenger" at wrong locations. These actions are erroneously marked because they are not explored.
- We observed that for PICKUP tasks the agent always chooses primitive actions but its interesting to note that in drop off tasks the agent irrespective of its current location, it chooses the option that leads the agent to the desired drop off destination.For Example Figure 13,last row last column.

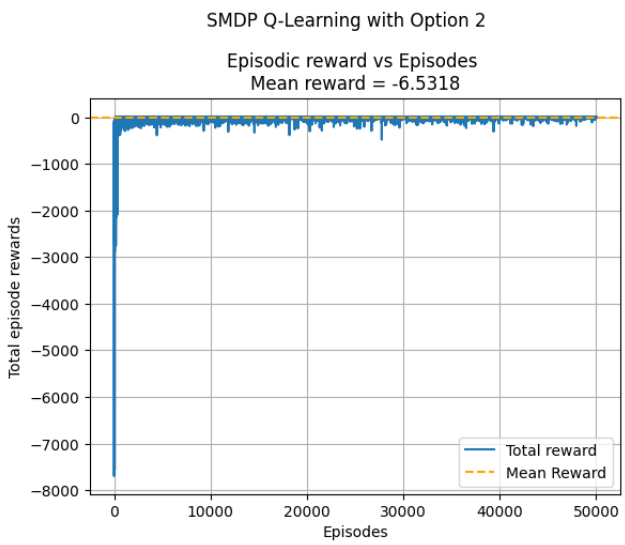
Reasoning: Why Intra-Option Q-Learning learns the policy:

- The reason behind the shift in behavior lies in the relatively high reward associated with dropping off a passenger (+20), which reduces the necessity for continued exploration post-pickup.
- The penalties incurred for each step (-1) and for picking up the passenger (-1) incentivize exploration.
- The discrepancy in exploration patterns is attributed to the need for the agent to explore more extensively before picking up a passenger compared to afterward.
- Occasional anomalies in the learned policy may occur, such as suggesting to pick up passengers at wrong locations. This happens when the corresponding Q value is zero, indicating a lack of exploration as their state-action values are higher than others in unexplored states or negative if explored.

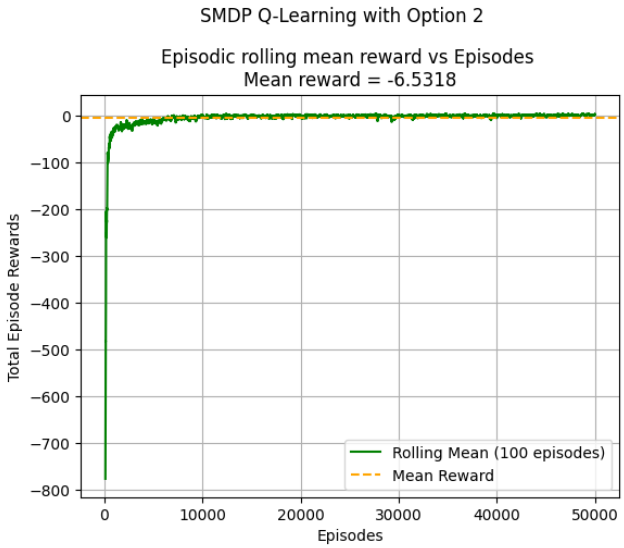
6 SMDP Q-Learning and Intra-Option Qlearning for OPTION SET 2

6.1 SMDP Q-Learning with Option Set 2

6.1.1 Reward curves and Visualization of learned Q-values

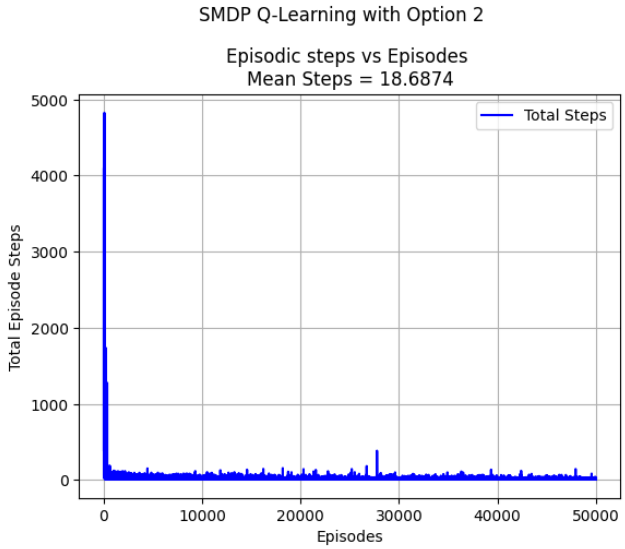


(a) Episodic Reward vs Episodes

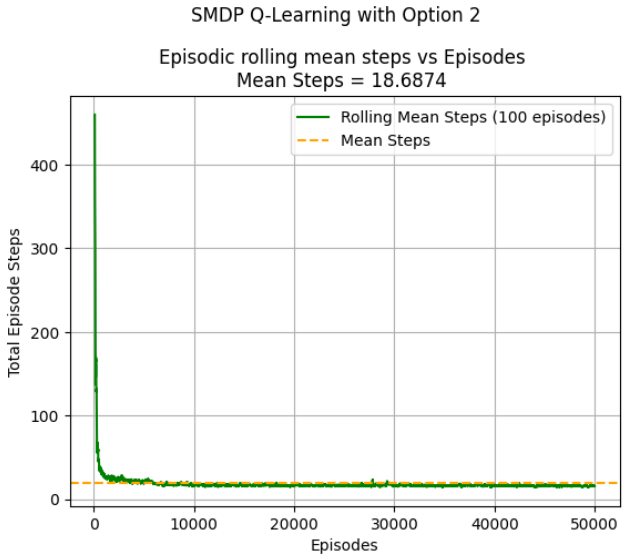


(b) Rolling mean Episodic Reward vs Episodes

Figure 17: Reward Plot for SMDP Q-Learning with Option Set 2



(a) Episodic Steps vs Episodes



(b) Rolling mean Episodic Steps vs Episodes

Figure 18: Steps Plot for SMDP Q-Learning with Option Set 2

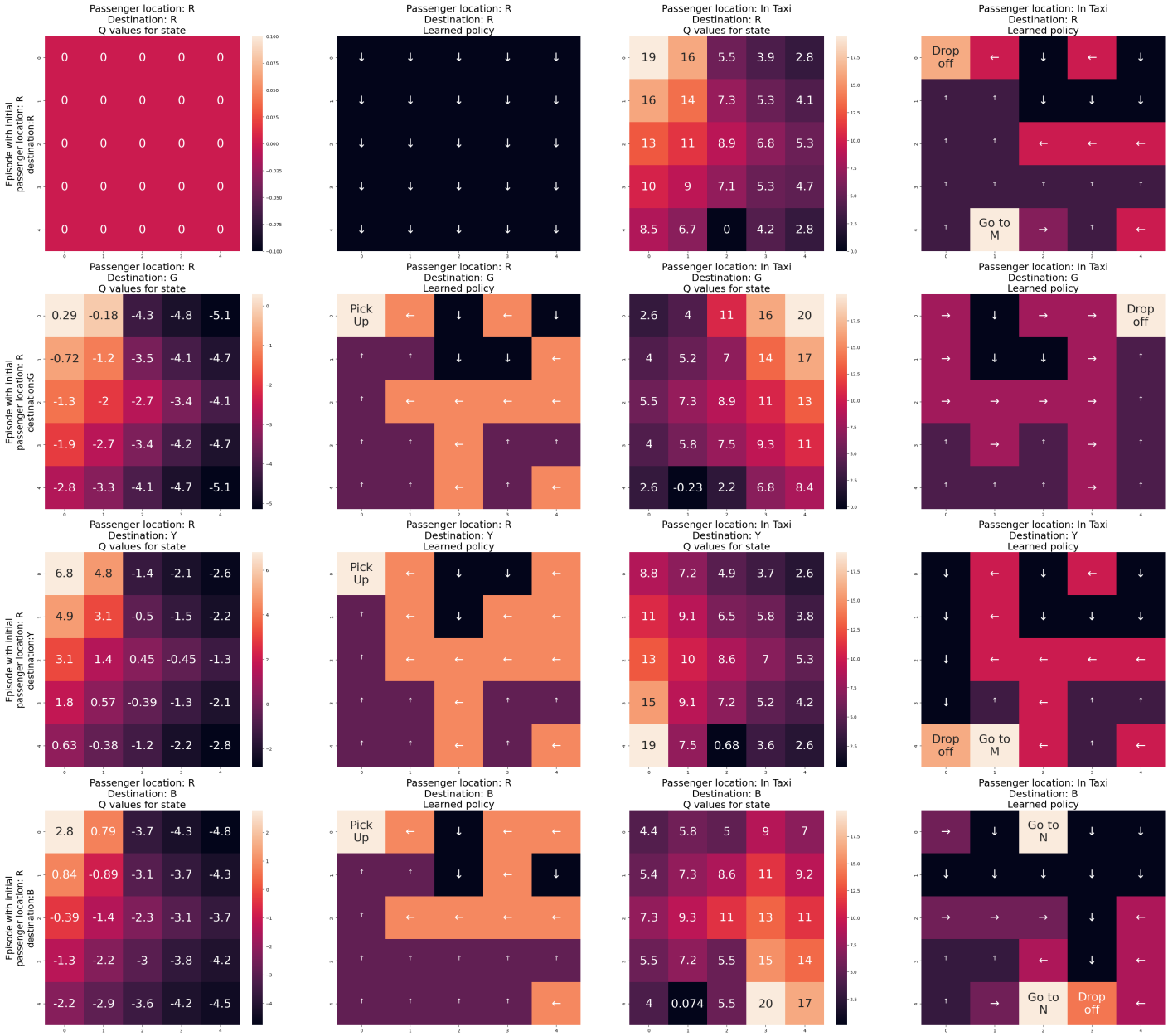


Figure 19: Learned Q-Values for SMDP Q-Learning with Option Set 2 for Passenger Location : R

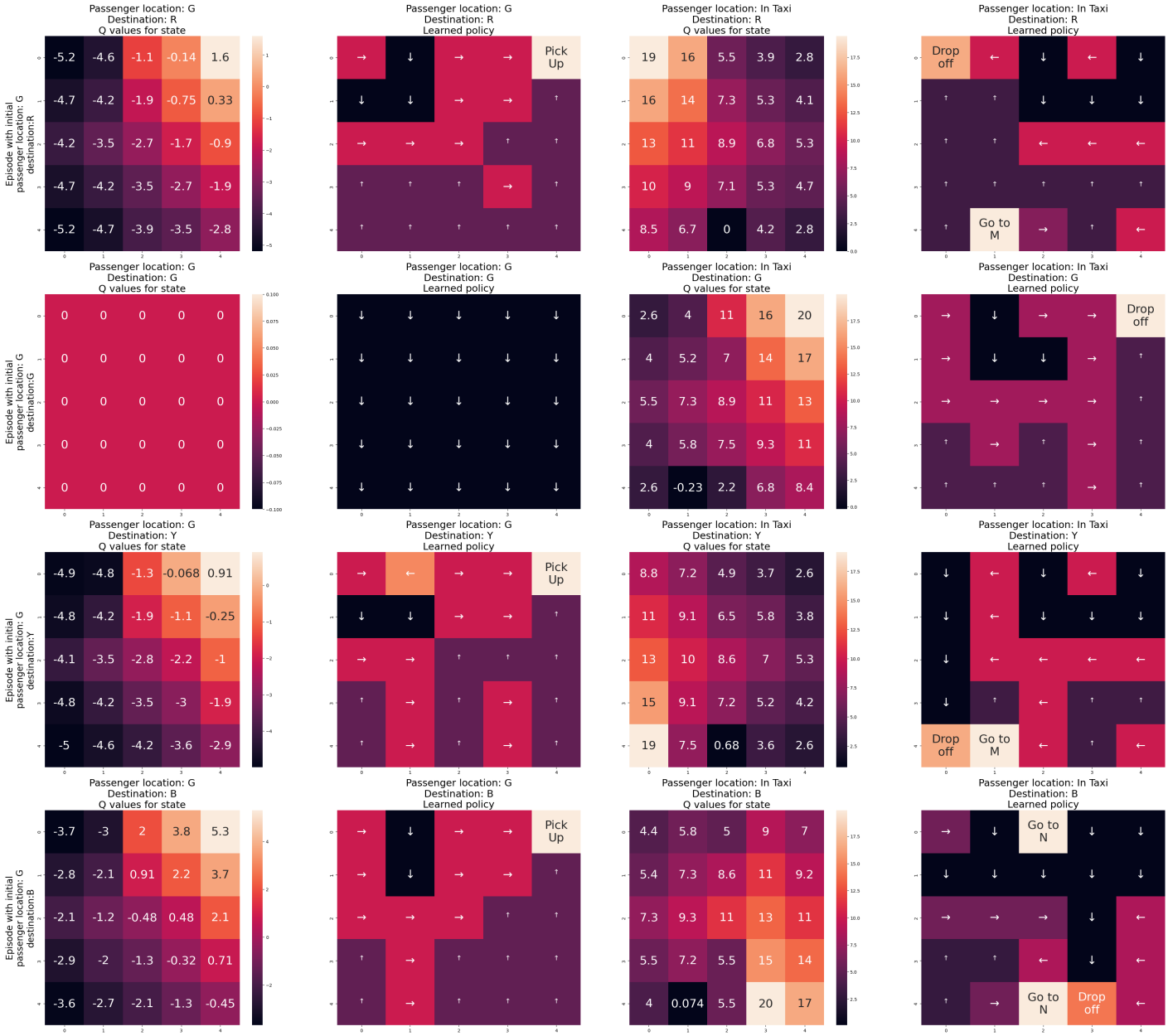


Figure 20: Learned Q-Values for SMDP Q-Learning with Option Set 2 for Passenger Location : G

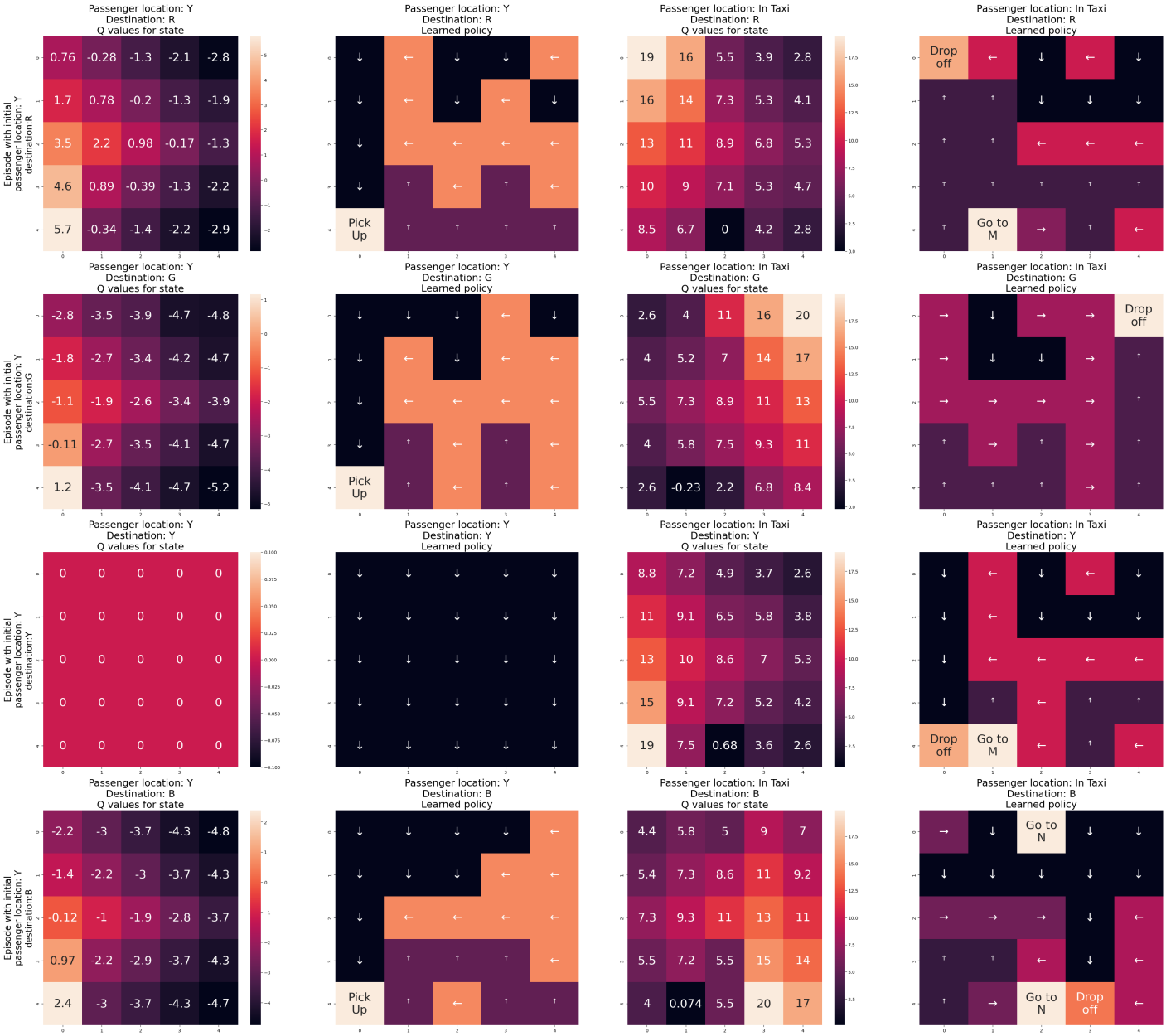


Figure 21: Learned Q-Values for SMDP Q-Learning with Option Set 2 for Passenger Location : Y

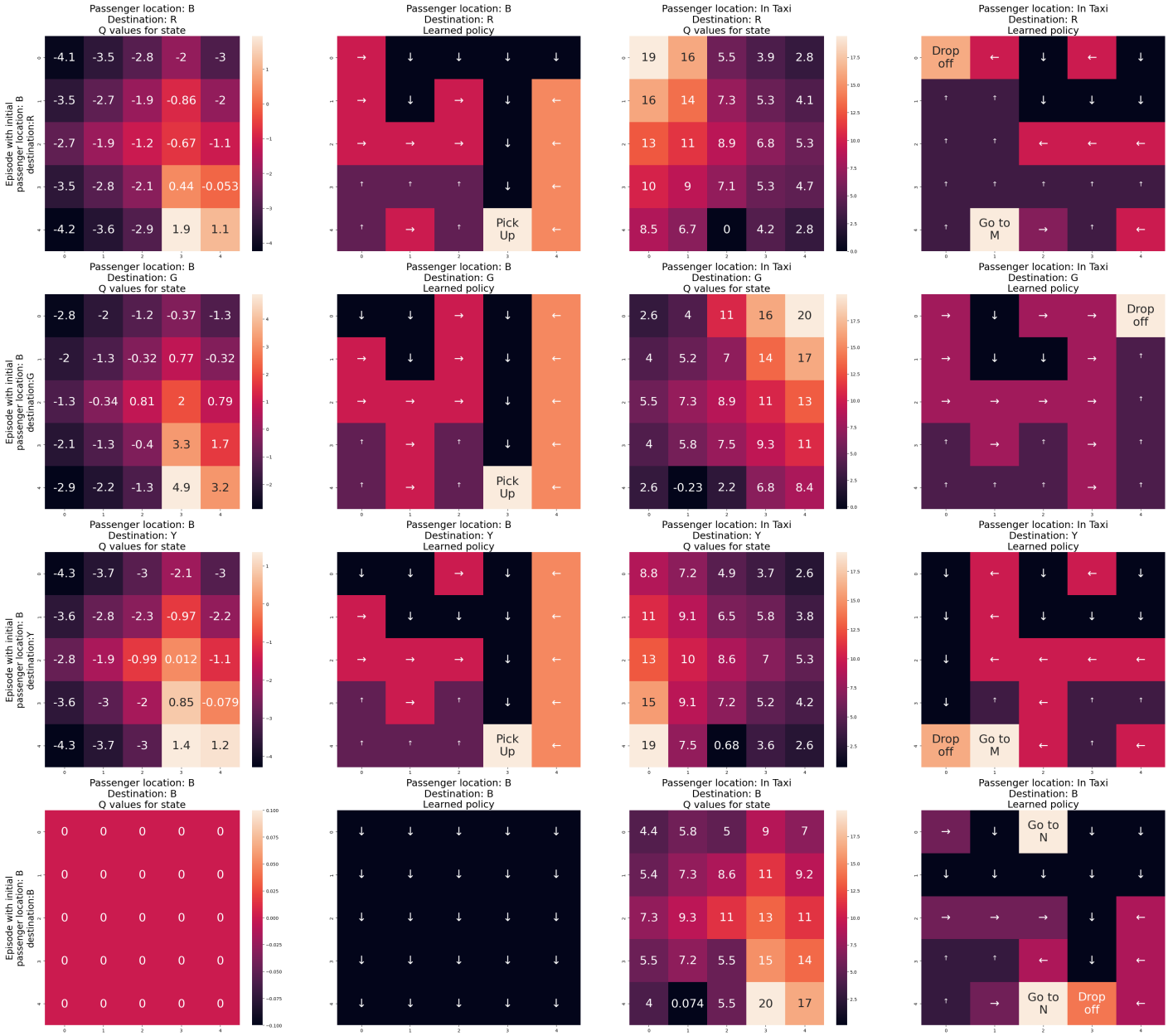


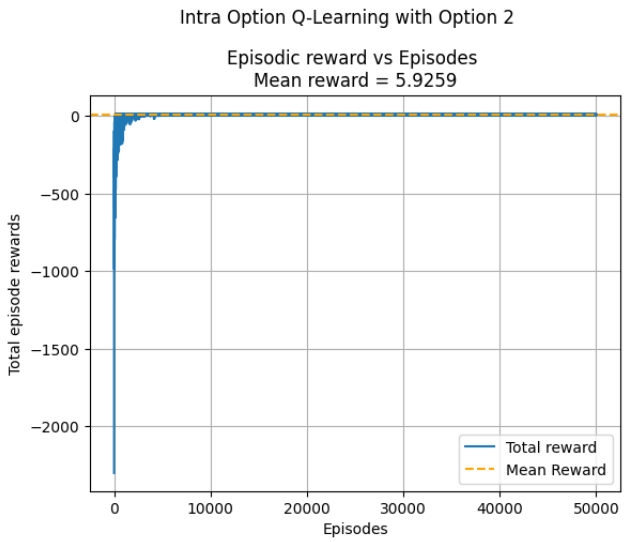
Figure 22: Learned Q-Values for SMDP Q-Learning with Option Set 2 for Passenger Location : B

6.1.2 Inferences from Learnt Policies((Comparing SMDP Option Set 1 vs SMDP Option set 2)

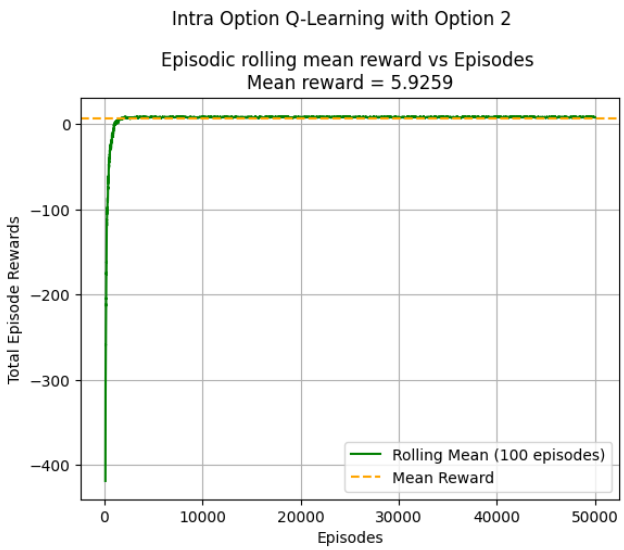
- We observed that for PICKUP tasks the agent always chooses primitive actions but its interesting to note that in DROP tasks the option which are closer to the destination.are picked by agent.
- Besides the unreachable states, there's a noticeable decrease in state-action pairs with zero value.
- To approach the optimal policy, the agent explores more state-action pairs, as emphasized in previous observations.
- The current options constrain the agent's movement to the left or right center of the grid, prompting exploration to reach the destination.
- This strategic approach has proven successful, as the agent now explores nearly the entire state-action space.
- Analysis of plots from Figure 22 , first row last column and last row last column reveals a tendency in the learned policy to prioritize manually-crafted options over primitive actions in many states.

6.2 Intra-Option Q-Learning with Option Set 2

6.2.1 Reward curves and Visualization of learned Q-values

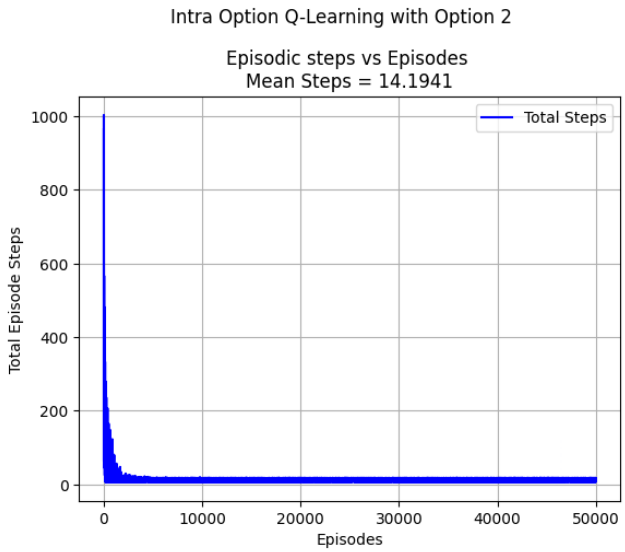


(a) Episodic Reward vs Episodes

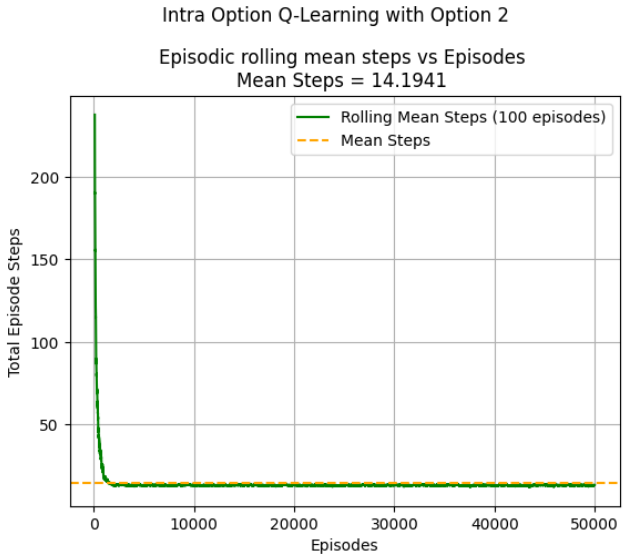


(b) Rolling mean Episodic Reward vs Episodes

Figure 23: Reward Plot for IOQL Q-Learning with Option Set 2



(a) Episodic Steps vs Episodes



(b) Rolling mean Episodic Steps vs Episodes

Figure 24: Steps Plot for IOQL Q-Learning with Option Set 2

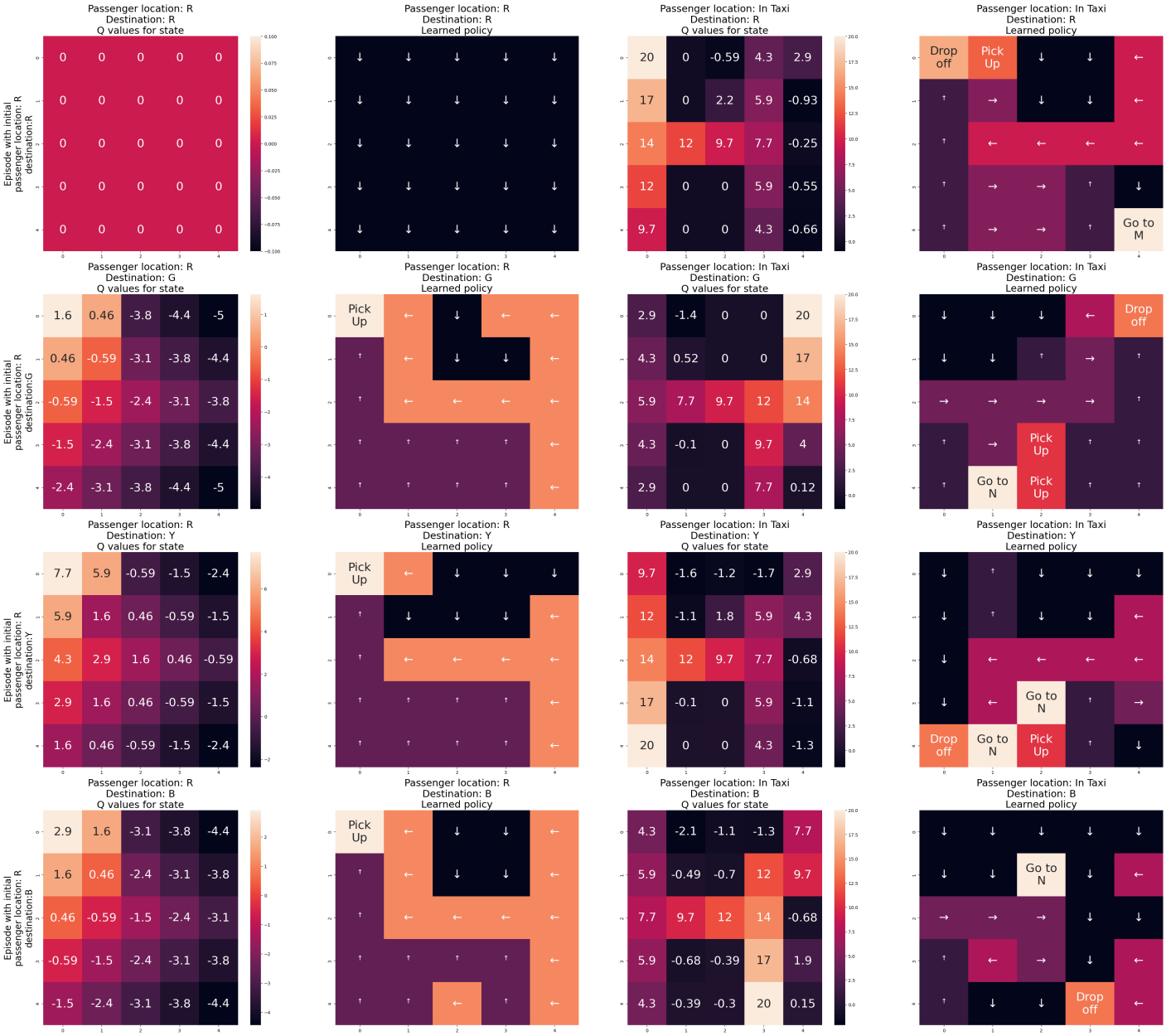


Figure 25: Learned Q-Values for IOQL Q-Learning with Option Set 2 for Passenger Location : R

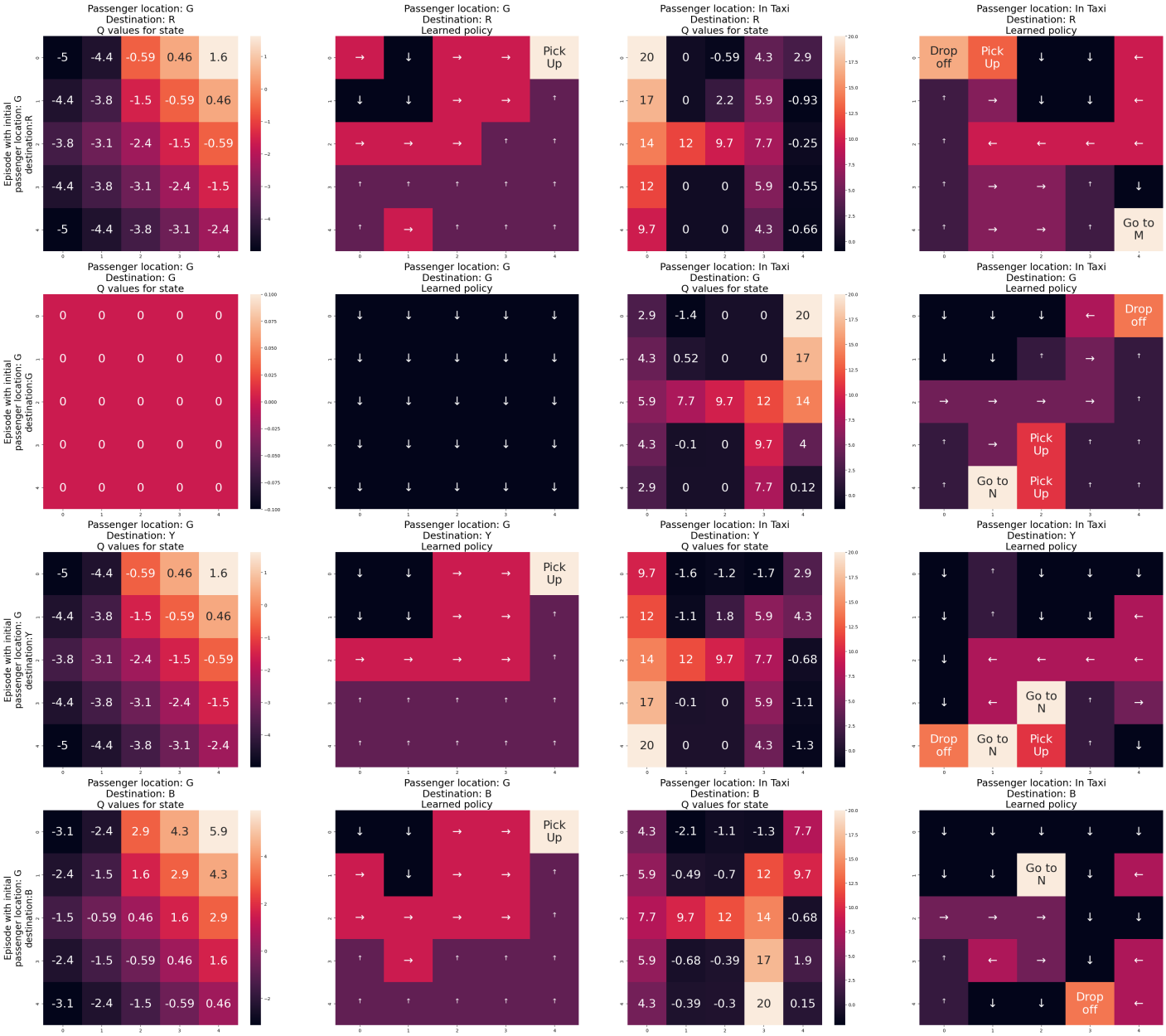


Figure 26: Learned Q-Values for IOQL Q-Learning with Option Set 2 for Passenger Location : G

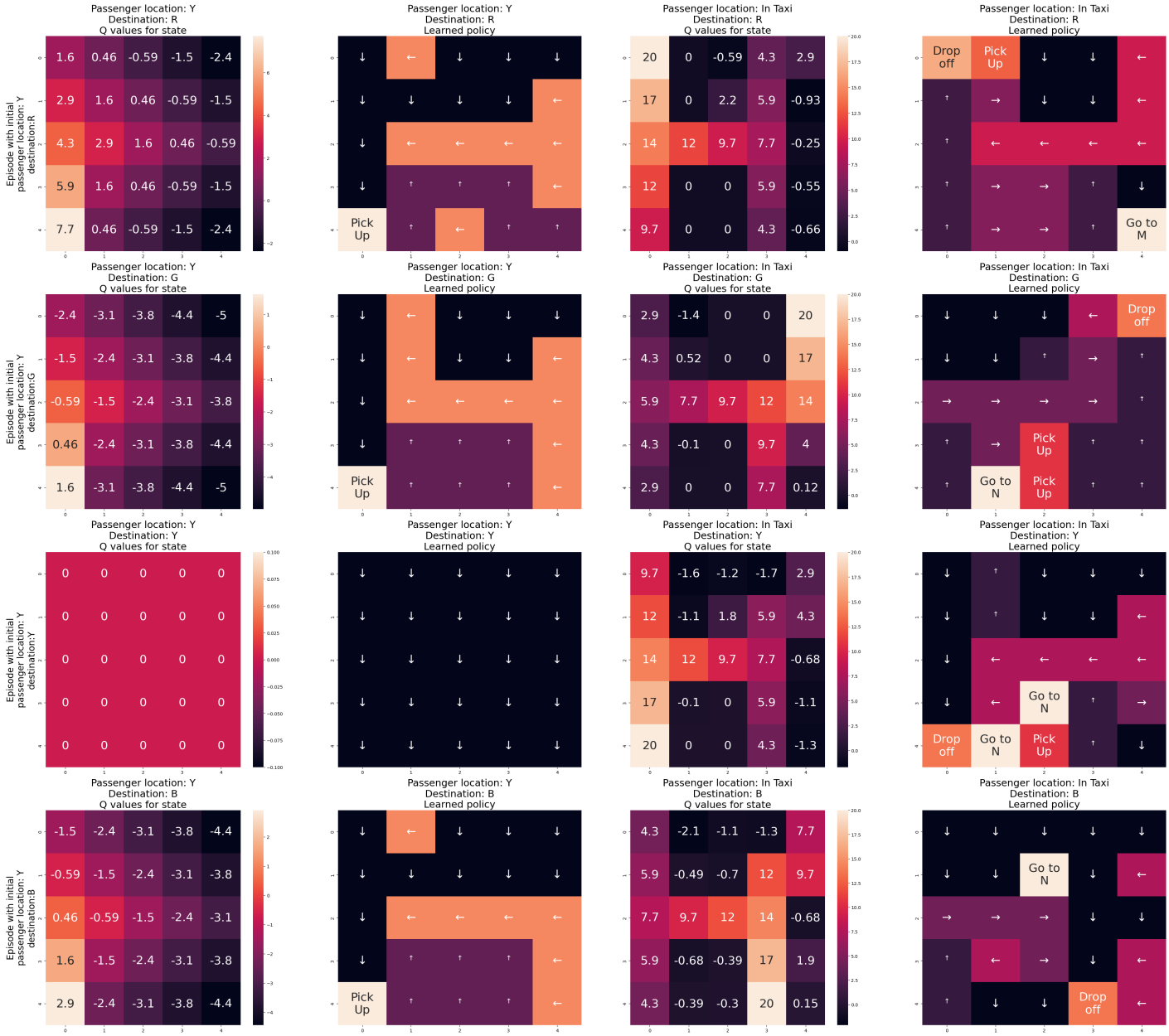


Figure 27: Learned Q-Values for IOQL Q-Learning with Option Set 2 for Passenger Location : Y

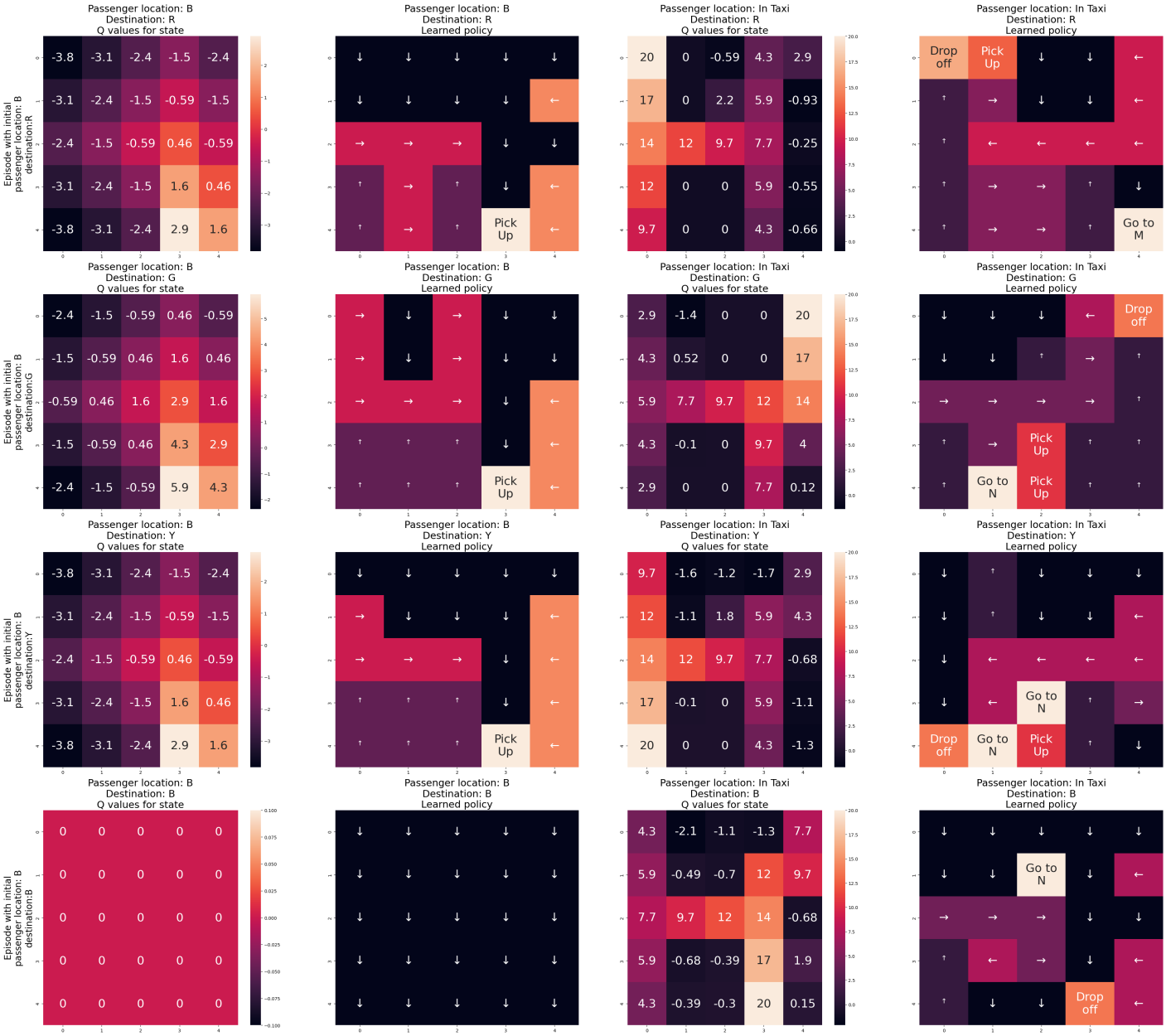
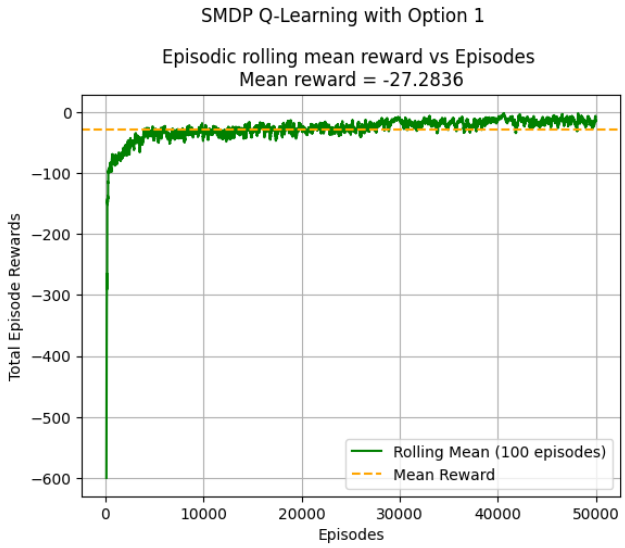


Figure 28: Learned Q-Values for IOQL Q-Learning with Option Set 2 for Passenger Location : B

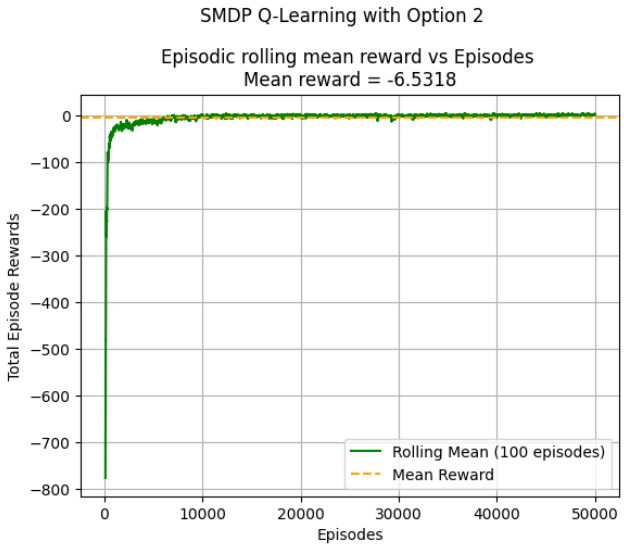
6.2.2 Inferences from Learnt Policies(Comparing IOQL Option Set 1 vs IOQL Option set 2)

- While certain cells in the plots may indicate "pick up passenger" or "drop off passenger" at wrong locations. These actions are erroneously marked because they are not explored.
- The plots suggests that every state-action pair has been explored, except for the unreachable states. This is corroborated by the absence of states with a value of 0 in the state value plots from Figure 25-28.
- The Q plots indicate that the learned policy is likely optimal or very close to optimal for PICKUP tasks But there are some anomalies for DROP off tasks especially for Y location(Figure 28 3rd row last column).
- This preference for options over primitive actions in certain states can be attributed to the substantial positive reward obtained from successfully dropping off the passenger. This reduces the agent's motivation to explore further.
- To address this, the policy could be refined to solely recommend primitive actions by either increasing the number of algorithm episodes or adjusting the reward system to promote continued exploration even after picking up the passenger.

7 Performance of SMDP Q-Learning and Intra-option Q-Learning algorithms with option set 1 and 2



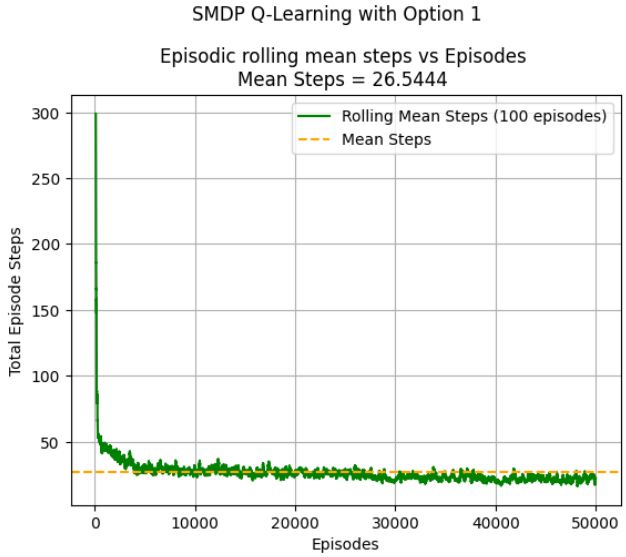
(a) Episodic Reward vs Episodes



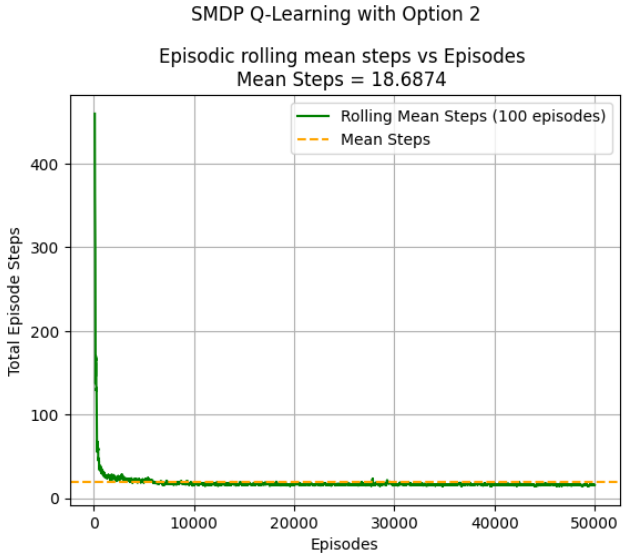
(b) Rolling mean Episodic Reward vs Episodes

Figure 29: Reward Plot for SMDP Q-Learning with Option Set 1

In Figure 29, we observe that the mean reward for option 2 is considerably higher



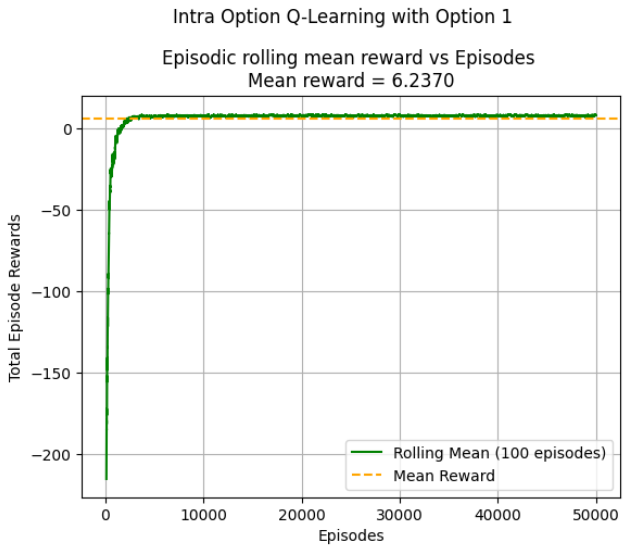
(a) Episodic Reward vs Episodes



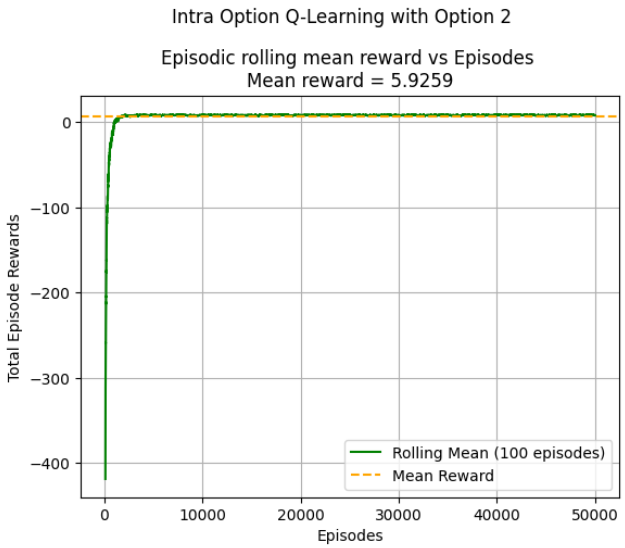
(b) Rolling mean Episodic Reward vs Episodes

Figure 30: Reward Plot for SMDP Q-Learning with Option Set 1

In Figure 30, we observe that the mean steps taken with set option 2 is lesser than option set 1



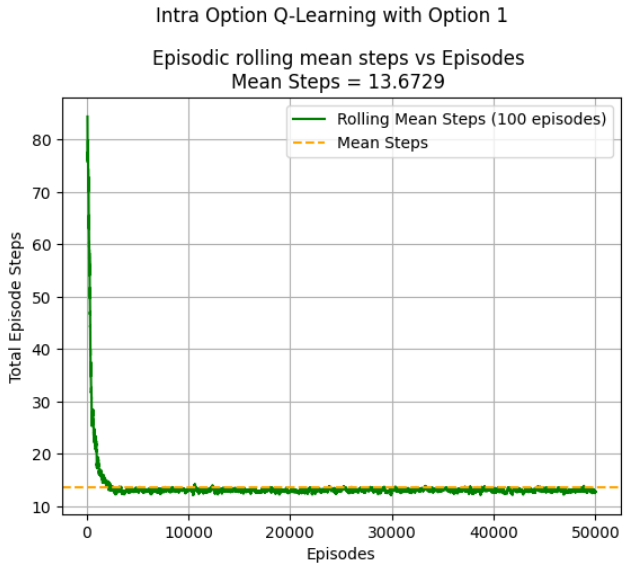
(a) Episodic Reward vs Episodes



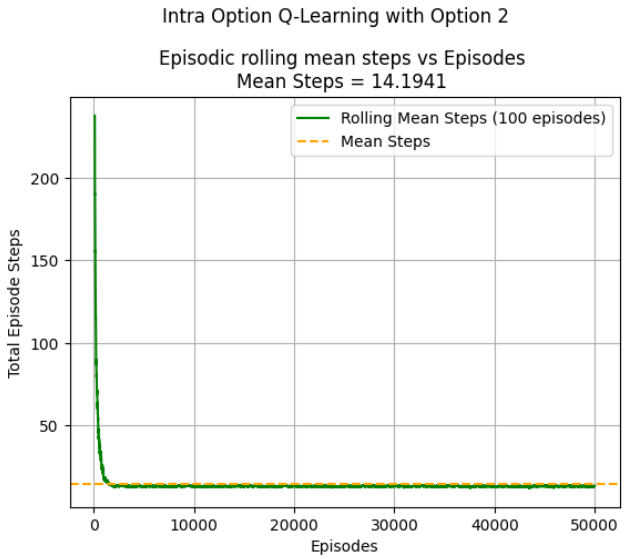
(b) Rolling mean Episodic Reward vs Episodes

Figure 31: Reward Plot for SMDP Q-Learning with Option Set 1

In Figure 31, we observe that the mean reward both are nearly same



(a) Episodic Reward vs Episodes



(b) Rolling mean Episodic Reward vs Episodes

Figure 32: Reward Plot for SMDP Q-Learning with Option Set 1

In Figure 32, we observe that the mean steps both are nearly same

8 Comparison between the SMDP Q-Learning and Intra-option Q-Learning algorithms

1. Convergence

SMDP Q-Learning:

- Treats options as "black boxes." Only updates the Q-value of the entire option when it finishes.
- This simplifies learning and reduces the number of updates needed,Although we observed from figure 30 and 32, Intr option converges in lesser number of steps.

Intra-Option Q-Learning:

- Updates both the Q-value of the entire option and the Q-values of individual state-action pairs within the option.
- Generally it provides more fine-grained learning, but requires more updates, potentially slowing down convergence compared to SMDP Q-learning, But in this environment we observed with tuned hypermaters that IO agent explores more number of state-action pairs which helps to converge the overall goal optimally .

2. Exploration

SMDP Q-Learning:

- Limited exploration within options. Only updates for the chosen option, neglecting other potential actions within it.
- May miss valuable discoveries if alternative actions within the option lead to better outcomes.

Intra-Option Q-Learning:

- Encourages exploration by updating state-action values during option execution.
- Learns the value of individual actions within the option, allowing the agent to discover better sub-paths within the option itself.

3. Action Learning

SMDP Q-Learning:

- Focuses on learning the overall value of options.
- The resulting policy might suggest entire options over individual actions in some situations, even if specific actions within those options are less desirable.

Intra-Option Q-Learning:

- Learns the value of individual actions within options.
- The resulting policy can leverage these learned action values. The agent can choose from a wider range of actions, including primitive actions within options, leading to potentially more nuanced and efficient behavior.

General observations:

- Policy learned by SMDP might be suboptimal due to limited exploration within options.
- With tuned hyperparameters,Intra-option Q-learning takes converges faster and performs better by exploring and learning individual actions within options.
- Comparing the learned policy plots of SMDP Q-learning with the Intra-option Q-learning, it's evident that Intra-option Q-learning offers primitive actions for each state for PICKUP tasks, whereas SMDP Q-learning suggests using manually-crafted options over primitive actions in certain states(Figure 8 third row second column,Figure 7 second row second column).
- The discrepancy between Intra-option Q-learning and SMDP Q-learning arises due to the off-policy nature of Intra-option Q-learning, which allows it to learn Q-values for actions and options that may not have been explored previously by updating the Q-value.