DA6400 - Reinforcement Learning

Programming Assignment 3

Patel Shrey Rakeshkumar (ME21B138)
Param Jivrajani (PH21B006)

Table of Contents

1 Introduction

2 Environment : Taxi-v3

Taxi-v3 is a part of the Toy Text environments. It involves navigating to passengers in a grid world, picking them up and dropping them off at one of four locations.
There are four designated pick-up and drop-off locations (Red, Green, Yellow and Blue) in the 5x5 grid world. The taxi starts off at a random square and the passenger at one of the designated locations. The goal is move the taxi to the passengers location, pick up the passenger, move to the passengers desired destination, and drop off the passenger. Once the passenger is dropped off, the episode ends. The player receives positive rewards for successfully dropping-off the passenger at the correct location. Negative rewards for incorrect attempts to pick-up/drop-off passenger and for each step where another reward is not received.

2.1 Description of the Environment

- **Action Space** : Consists of an ndarray with shape (1,) with 5 discrete actions, namely
  - 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** : The Taxi environment consists of 500 discrete states, computed as:

$$(\text{taxi\_row} \times 5 + \text{taxi\_col}) \times 5 + \text{passenger\_location}) \times 4 + \text{destination}$$

  - 25 taxi positions: 5×5 grid
  - 5 passenger locations: Red, Green, Yellow, Blue, In taxi
  - 4 destination locations: Red, Green, Yellow, Blue

  Only 400 of these states are reachable in typical episodes. Some states (e.g., passenger at destination) are terminal and not revisited. Including 4 post-success states, there are 404 effectively reachable states.
- **Starting State** : The initial state is sampled uniformly from the possible states where the passenger is neither at their destination nor inside the taxi. There are 300 possible initial states: 25 taxi positions, 4 passenger locations (excluding inside the taxi) and 3 destinations (excluding the passengers current location).
- **Rewards**
  - -1 per step by default.
  - $+20$ for successfully delivering the passenger.
  - -10 for illegal "pickup" or "drop-off" actions.

Invalid actions (e.g., moving into walls) incur a penalty and can be avoided using the `action_mask` from `info`.

- **Episode End**
  - **Termination:** Passenger is successfully dropped off.
  - **Truncation:** Episode exceeds 200 steps.

3 Implementation and Code Snippets

3.1 Explanation of Code and Options

In the following code, we have used Hierarchical Learning using Options for learning the environment. The two algorithms we used are **SMDP** (*Semi Markov Decision Process*) and **Intra-Option Q-Learning**.
Each option is defined as a triple : (I, π_0, β). I is the initiation set, π_0 is policy followed during the option and β is the probability of the option terminating in that set.
The options are defined in advance and are fixed. While running each episode, **we treat each option as an option** and define the Q-table in the combined space of options and primitive actions. We choose an action/option from this combined space using a policy (ϵ-*greedy* or *softmax*).

For each algorithm, the update rules and the explanation of our implementation are given below :

- **SMDP** : A global Q-table is maintained in the combined space of actions and options. While running the episode, an action/option is chosen from this space using the Global Q-table. If a primitive action is chosen, then the update is performed using the usual 1-step TD-Update :

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left[r_t + \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right] \tag{1}$$

  If an option o is selected, the option is run till termination. No state action values in the Global table are updated until the option terminates.

  For each option, we maintain a separate Q-table with shape (5,5,primitive_actions) for each option. While running the option, these values are used to select actions rather than the global Q values, using the . We also maintain a separate ϵ for each option. For SMDP, the option is run until termination. The rewards are discounted at each step inside the option. When the option is terminated, the Global Q values are updated using the update :

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left[\bar{r}_{t+\tau} + \gamma^{\tau} \max_{a'} Q(s_{t+\tau}, a') - Q(s_t, a_t) \right] \tag{2}$$

  where $\bar{r}_{t+\tau}$ is the total discounted reward inside the option.

  However, the Q-table for each option is updated during each step of the option, using the 1-step Q-learning update.

- **Intra-Option Q-Learning** : A global Q-table is maintained same as in SMDP. While running the episode, the action/option is chosen using the policy over the global Q table. If a primitive action is chosen, the update is performed using the 1-step update shown in (1). The update is different however, for the options. Separate Q-tables are maintained for each option. When the option is running, the primitive actions are chosen using these Q-tables. The option policies

have their own parameters. The update while running the policy is different than SMDP. Unlike SMDP, where the global Q parameters are updated after the option is terminated, here, the global Q-values are updated during each step of the option along with the Q-tables for the option.

Moreover, we also update the Q-table for all the options apart from the current option which would have taken the same action under the policy.

The above updates are summarized below:

$$\begin{aligned} Q(s_1, o) &= Q(s_1, o) + \alpha \left[r_1 + \gamma Q(s_2, o) - Q(s_1, o) \right] && \text{If not terminating at } s_2 \\ &= Q(s_1, o) + \alpha \left[r_1 + \gamma \max_a Q(s_2, a) - Q(s_1, o) \right] && \text{If terminating at } s_2 \end{aligned}$$

Suppose $\pi_o(s_1) = \text{a}$ & $\pi_{o'}(s_1) = \text{a}$

When executing option o, option o' can also be updated

$$Q(s_1, o') = Q(s_1, o') + \alpha \left(r_1 + \gamma Q(s_2, o') - Q(s_1, o') \right)$$

3.2 Code Snippets

- The state is given as a number between 1 to 500 which is decoded to give a tuple (row,column,passenger location,destination). The state is decoded into the tuple using the following code:

```
def decode(self, state):
    taxi_row = state // 100
    taxi_column = (state // 20) % 5
    passenger_location = (state % 20) // 4
    destination = (state % 20) % 4
    return (taxi_row, taxi_column, passenger_location, destination)
```

- The policies are defined as follows :

```
def get_action(self, state_idx):
    actions = self.action_shape + self.option_shape
    if self.policy == 'epsilon_greedy':
        if np.random.random() < self.param:
            return np.random.randint(0, actions)
        else:
            return np.argmax(self.Q[state_idx][:actions])
    elif self.policy == 'softmax':
        q_values = self.Q[state_idx][:actions] / self.param
        q_values = q_values - np.max(q_values)
        exp_values = np.exp(q_values)
        probabilities = exp_values / np.sum(exp_values)
        return np.random.choice(actions, p=probabilities)
```

- There are two sets of options used here. The first one is defined using the passenger location, being at one of the four colours Red, Blue, Green, Yellow. The second is defined in the next section, where pickup or dropoff is performed based on the location of the passenger being in the taxi or not.

```
def is_red(self, state_decode):
    return state_decode[-2] == 0

def is_green(self, state_decode):
    return state_decode[-2] == 1

def is_yellow(self, state_decode):
    return state_decode[-2] == 2

def is_blue(self, state_decode):
    return state_decode[-2] == 3
```

- The Option is executed as described above with the help of the following code :

```
elif self.algo =='smdp': self.update_opt(opt_id=opt_idx, state_idx = (taxi_row,
                                        taxi_col), action=optaction,reward=
                                        reward, terminated=terminated,
                                        next_state_idx=(next_taxi_row,
                                        next_taxi_col))
            self.update(state_idx = state, action = opt_id, reward = reward,
                                                terminated = terminated,
                                                next_state_idx =
                                                next_state)

elif self.algo == 'smdp':
    self.update_opt(opt_id=opt_idx, state_idx=(taxi_row, taxi_col), action=
                                            optaction,reward=reward,
                                            terminated=terminated,
                                            next_state_idx=(next_taxi_row,
                                            next_taxi_col))
```

- The episodes are run as follows :

```
reward_list = []

for i in tqdm(range(n_episodes), desc="Training"):
    state, _ = env.reset()
    done = False
    reward_TT = 0
    while not done:
        action = agent.get_action(state_idx=state)
        if action <= 5:
            next_state, reward, terminated, truncated, _ = env.step(action=
                                                action)
            agent.update(
                state_idx=state,
                action=action,
                reward=reward,
                terminated=terminated,
                next_state_idx=next_state
            )
            reward_TT += reward
        else:
            total_reward, undecayed_reward, steps, next_state, terminated,
                                                truncated = agent.Option(
                                                state, action)
            if agent.algo == 'smdp':
```

```
                future_q = 0 if terminated or truncated else np.max(agent.Q[
                                                        next_state])
                agent.Q[state][action] += agent.learning_rate * (
                    total_reward + (agent.discount_factor ** steps) * future_q -
                                                          agent.Q[state][
                                                        action]
                )
            reward_TT += undecayed_reward
        state = next_state
        done = terminated or truncated
    agent.decay()
```

4 Use of Options

We used two different sets of Options to learn the environment.

- **Type 1** :
  We check the location of the taxi and move it to each of the four locations Red, Blue, Green, Yellow. We check the x and y coordinates of the taxi and move the taxi to the designated locations.
- **Type 2** :
  We check the location of the passenger. If the passenger is inside the taxi, then we run the option of Dropping off the passenger to the destination. If the passenger not inside the taxi, then we execute the option of Picking up the passenger.

5 Values Used

- The parameters used in this code are :

Parameters	**Values**
Discount Factor	0.99
Algorithm	[`Intra_Option`, `SMDP`]
Option Choice	[1, 2]

Table 1: Parameters and their corresponding values

- The Hyperparameters used are :

Hyperparameters
Episodes
Experiments
Learning Rate
Policy (ϵ-greedy, softmax)
Parameter
Decay Parameter
Final Parameter

Table 2: List of Hyperparameters

6 Results and Plots

- Average reward and rolling mean of the average reward over the 5 experiments have been plotted for the best hyperparameter value for both the sets of options for Intra option and SMDP.
- For each option and for each algorithm, we also plot the heatmap for the options showing the actions on the 5x5 grid to visualize the policies and the Q-values for each square on the grid.
- The Cumulative mean regret is minimized during the sweeps to find the optimal values.
- The sweep is done over the values using *wandb*.

```
        import wandb
import yaml
import argparse
# Initialize sweep with data from .yaml file
with open("sweep.yaml", "r") as file:
    sweep_configuration = yaml.safe_load(file)
wandb.require("core")
# Getting arguments for entity and project name
parser = argparse.ArgumentParser(
    description="Training script that return model weights")
parser.add_argument(
    '-we',
    '--wandb_entity',
    type=str,
    default=None,
    help='Wandb Entity used to track experiments in the Weights & Biases
                                                dashboard')
parser.add_argument(
    '-wp',
    '--wandb_project',
    type=str,
    default=None,
    help='Project name used to track experiments in Weights & Biases dashboard')
parser.add_argument(
    '-c',
    '--count',
    type = int,
    default=100,
    help = 'Maximum number of sweep per agent')
args = parser.parse_args()
sweep_id = wandb.sweep(
    sweep_configuration,
    entity=args.wandb_entity,
    project=args.wandb_project
)
# Initilizing agent
wandb.agent(sweep_id, count=args.count)
```

6.1 SMDP

The results and plots for the best hyperparameter values for each set of options are given.

6.1.1 Option Type 1

Hyperparameters	Values
Episodes	10000
Experiments	5
Learning Rate	0.493
Policy	softmax
Parameter	0.251
Decay Parameter	0.940
Final Parameter	0.012

Table 3: Hyperparameters and their Values

- Cumulative Mean Regret = 8.961
- Cumulative Mean Reward = -8.961

Figure 1: Destination Red and Green

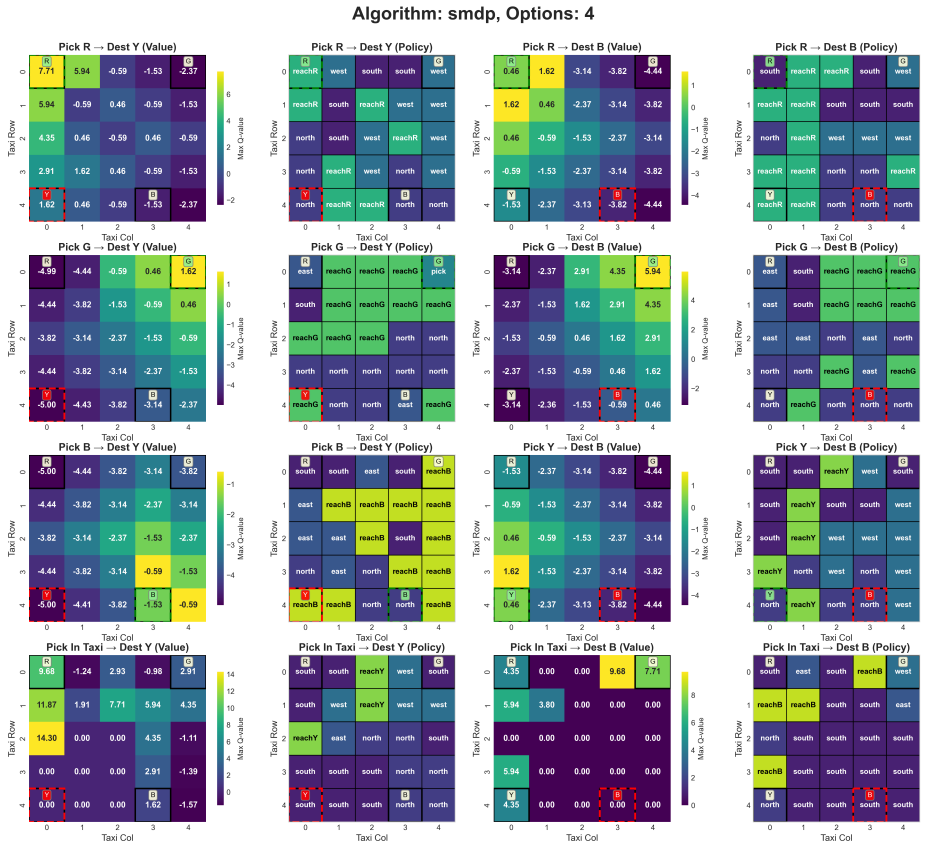


Figure 2: Destination Yellow and Blue

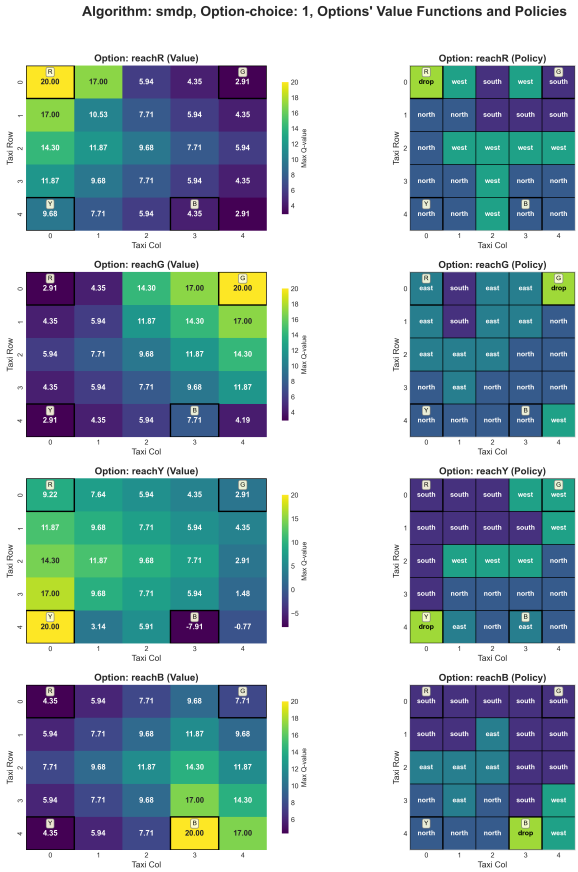


Figure 3: Option Policy

6.1.2 Option Type 2

Hyperparameters	Values
Episodes	10000
Experiments	5
Learning Rate	0.49
Policy	softmax
Parameter	3.11
Decay Parameter	0.916
Final Parameter	0.052

Table 4: Hyperparameters and their Values

- Cumulative Mean Regret = 17.226
- Cumulative Mean Reward = -17.226

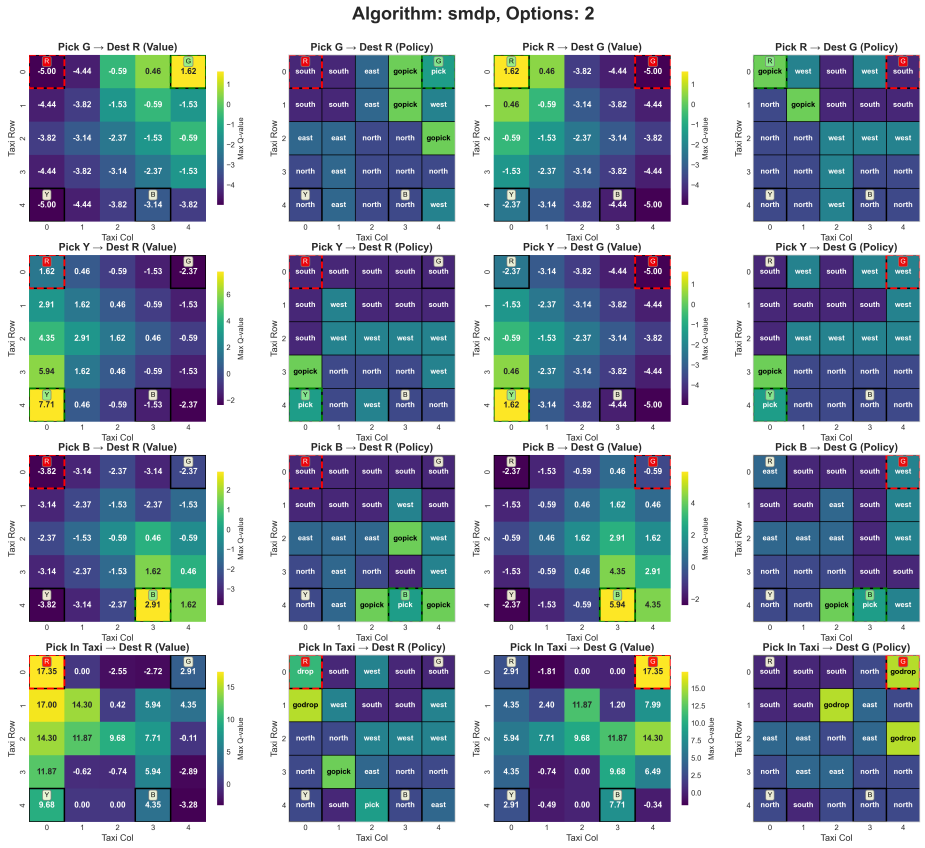


Figure 4: Destination Red and Green

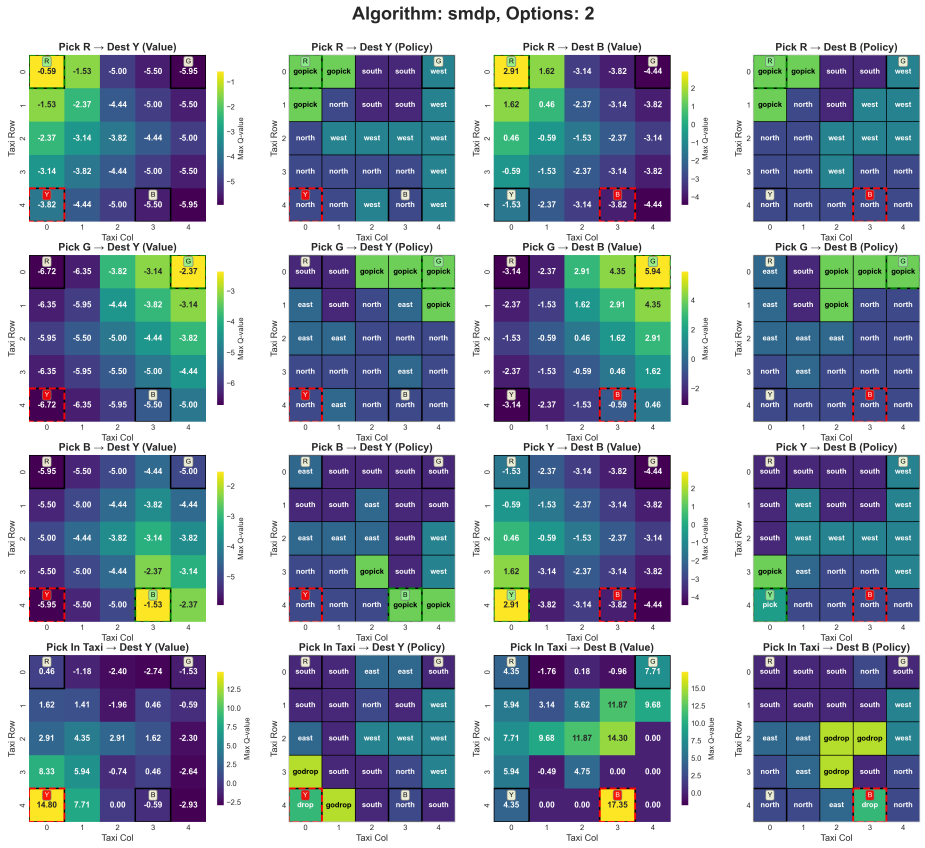


Figure 5: Destination Yellow and Blue

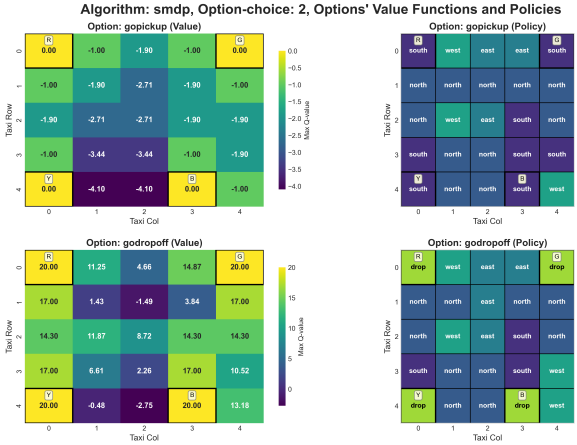
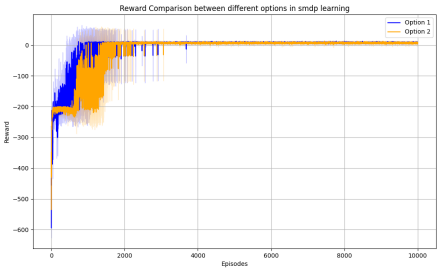
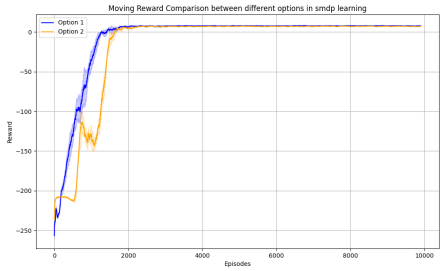


Figure 6: Option Policy



(a) Total Average Reward



(b) Rolling mean reward

Figure 7: SMDP

- SMDP learns the above given policies for both options. We can see that it chooses the options for a lot of grid squares, and we can see that this may be because the Q values are updated after accumulation.

6.2 Intra-Option Q-Learning

6.2.1 Option Type 1

Hyperparameters	**Values**
Episodes	10000
Experiments	5
Learning Rate	0.435
Policy	Softmax
Parameter	18.379
Decay Parameter	0.931
Final Parameter	0.054

Table 5: Hyperparameters and their Values

- Cumulative Mean Regret = 2.630
- Cumulative Mean Reward = -2.630

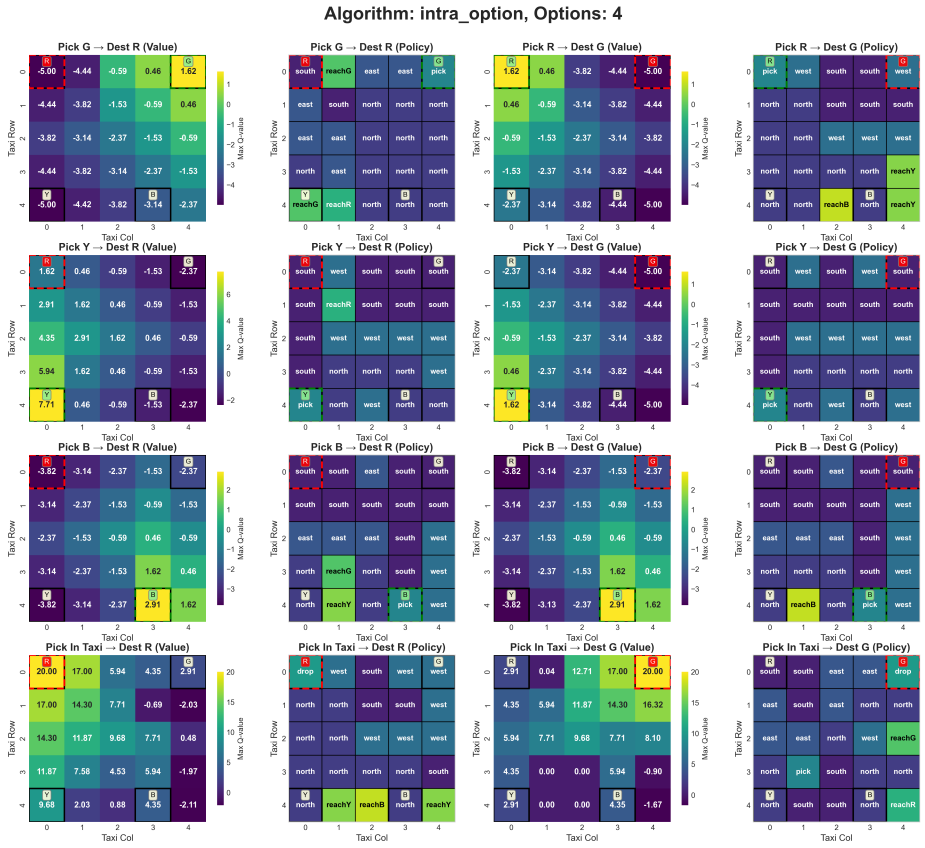


Figure 8: Destination Red and Green

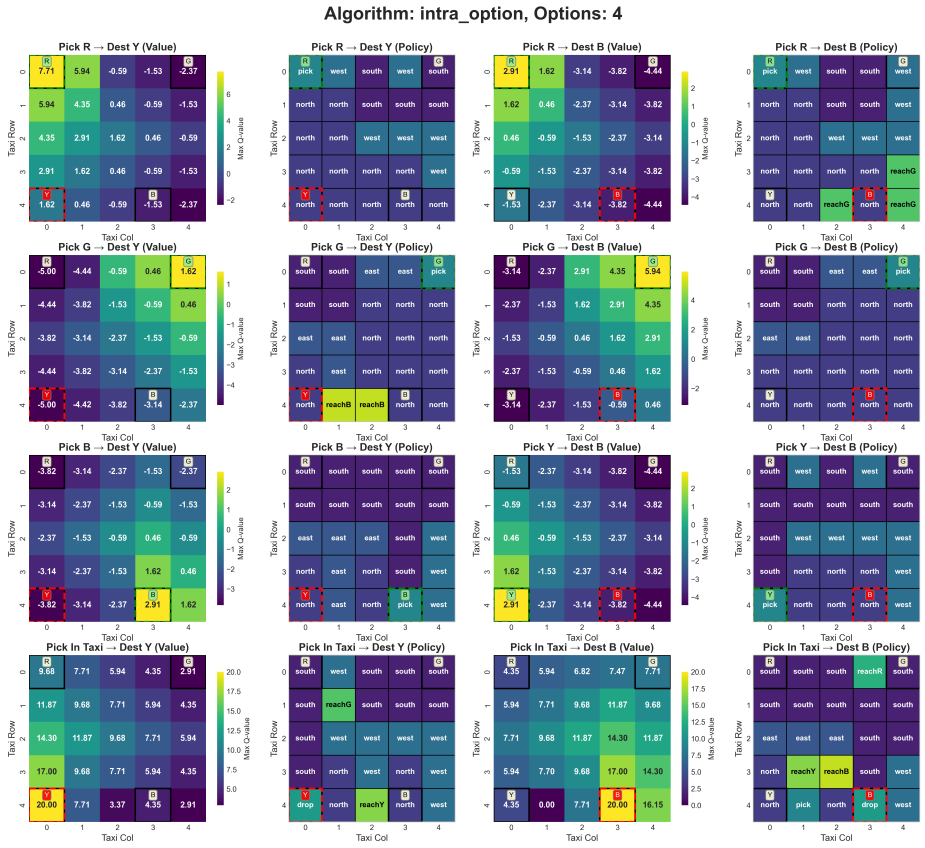


Figure 9: Destination Yellow and Blue

Figure 10: Option Policy

6.2.2 Option Type 2

Hyperparameters	Values
Episodes	10000
Experiments	5
Learning Rate	0.319
Policy	softmax
Parameter	76.743
Decay Parameter	0.961
Final Parameter	0.057

Table 6: Best Hyperparameter values

- Cumulative Mean Regret = 6.745
- Cumulative Mean Reward = -6.745

Figure 11: Destination Red and Green

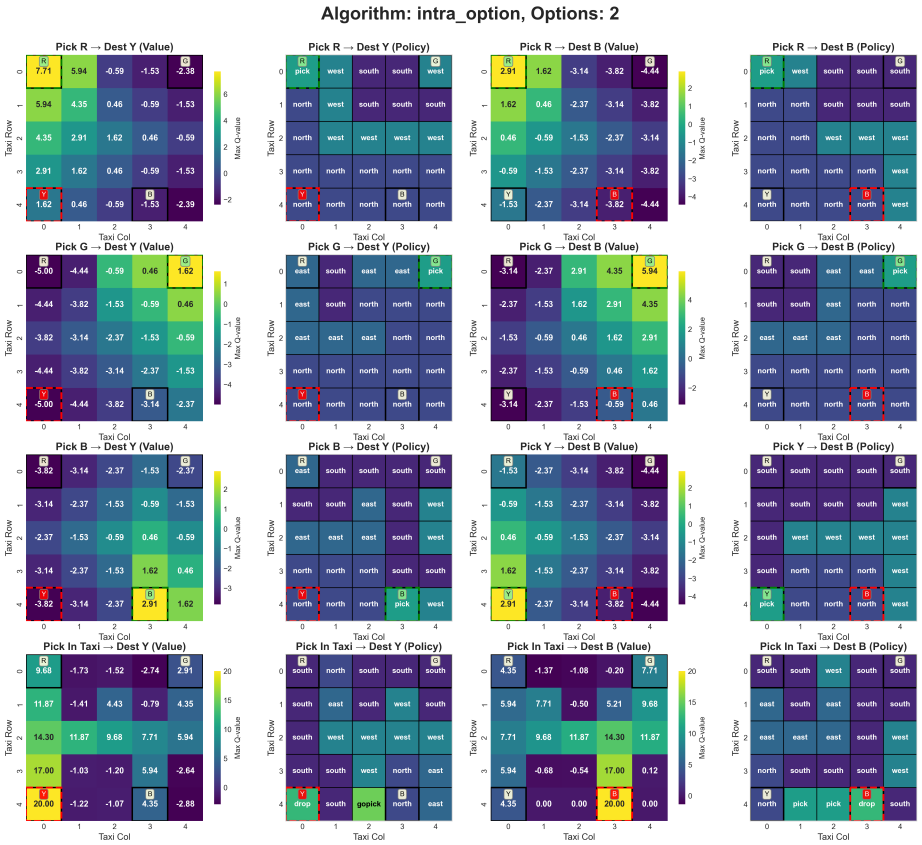


Figure 12: Destination Yellow and Blue

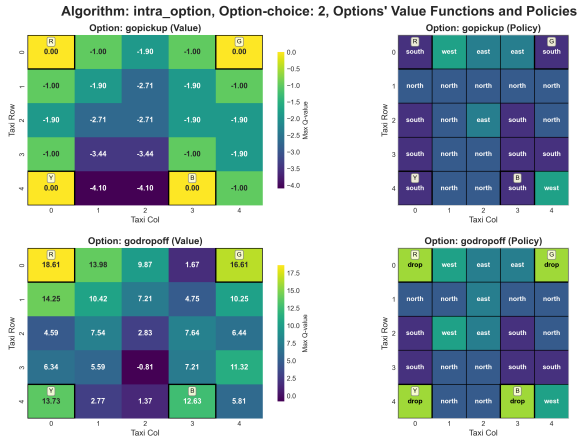
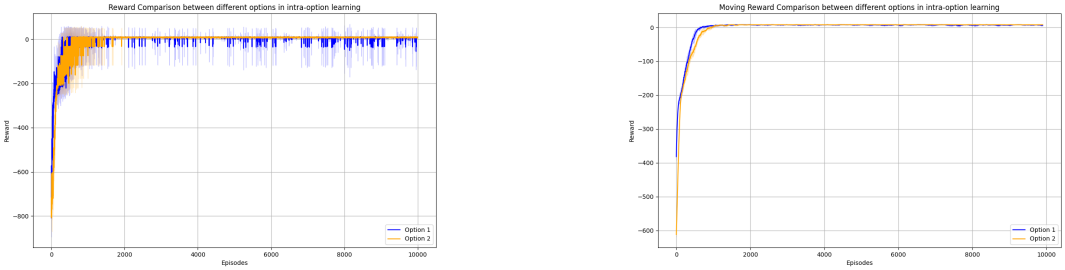


Figure 13: Option Policy



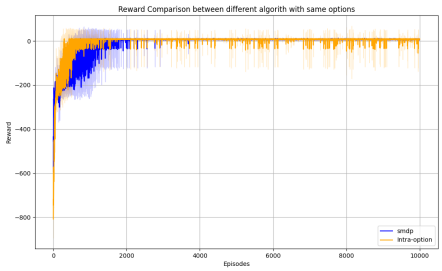
(a) Total Average Reward

(b) Rolling mean reward

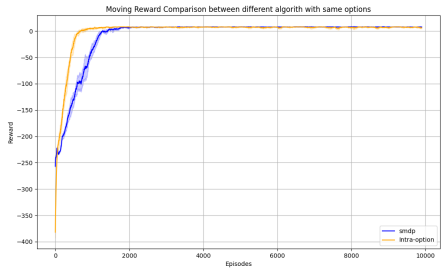
Figure 14: Intra-Option

6.3 Comparison Between SMDP and Intra-Option

6.3.1 Option Type 1



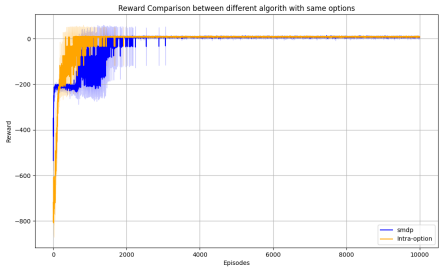
(a) Total Average Reward



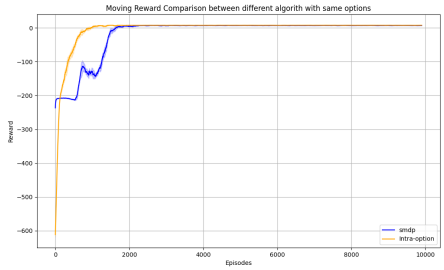
(b) Rolling mean reward

Figure 15: Option Type 1

6.3.2 Option Type 2



(a) Total Average Reward



(b) Rolling mean reward

Figure 16: Option Type 2

- Intra-Option Q-Learning learns the above policies for both options. We can see that it chooses the options for some of the states in the grid, and we can see that because the Q values are updated after each step inside the option, it picks the primitive actions as the best choice for comparatively fewer grid squares than SMDP.

6.4 Conclusions

- It can be observed from the plots and the Cumulative Mean Regrets that Intra-Option Q-learning performs better than SMDP in general. It converges to a mean reward of 0 much faster compared to SMDP with comparatively less variance.
- This can be because during Intra-option, the Q-values are updated while the option is running as compared to SMDP where the Q values are updated only after the option terminates.

7 Link to Github

The link to the github repo has been attached here.

8 References

- Gymnasium Documentation
- Wandb Documentation

9 Acknowlegements

We would like to thank Prof.Balaram Ravindran and the Teaching Assistants for clearing our queries during the assignment.