CS6700: Reinforcement Learning

Programming Assignment 3

Royyuru Sai Prasanna Gangadhar - ME19B190

Course Professor : Prof. Balaraman Ravindran

INDEX

April 25, 2023

Contents

List of Figures

1 Abstract

In this assignment we explore Semi Markov Decision Process (SMDP) Q learning and Intra option Q learning on "Taxi-v3" environment by opengym AI. We also use options and determine their usefulness in learning value functions and there by learning optimal policies.

The main aim of the assignment is to understand the SMDP Q learning and Intra option Q learning with options. We also analyze two mutually exclusive options and compare the configuration that gives better results.

In order to keep the report concise only important plots and tables have been added in this assignment. All the plots that have been obtained during the working of assignment have been stored in a folder and zipped together along with the code files and the report. Code snippets where ever necessary have been added in this report. The performance metric that has been used to compare various configurations is chosen to be the number of time steps that the agent takes to drop a passenger

2 Environment Description

- **Taxi-v3:** The environment for this task is the taxi domain, illustrated in Fig. 1. It is a 5x5 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. 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. 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)

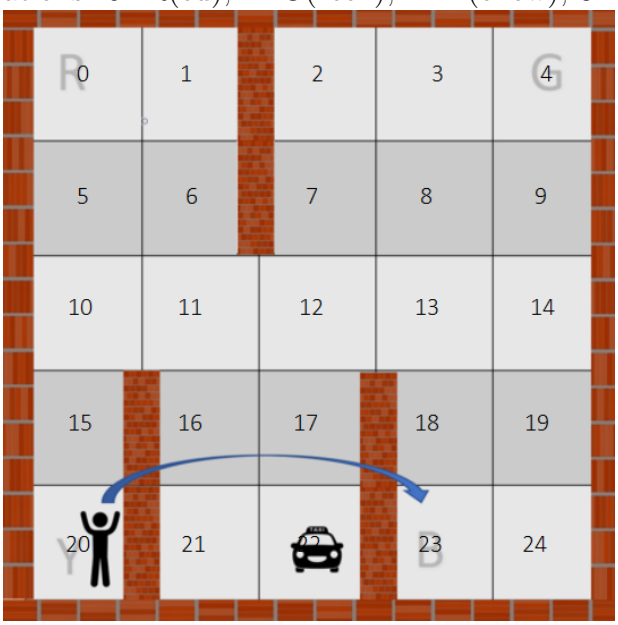

Figure 1: Visualizing Taxi-v3 environment with positions annotated

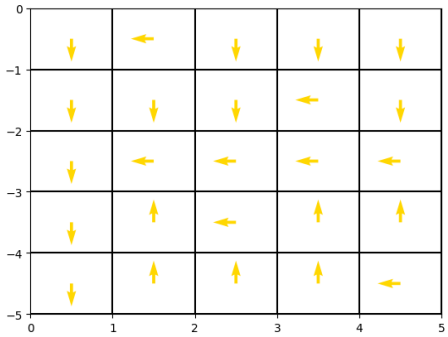

Figure 2: Policy to reach the state "Yellow"

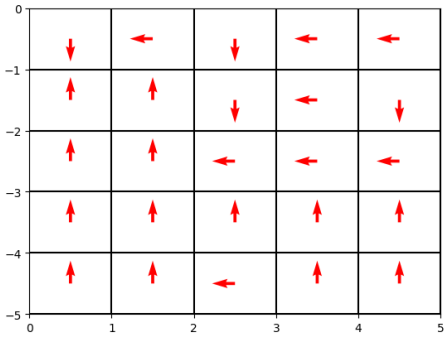

Figure 3: Policy to reach the state "Red"

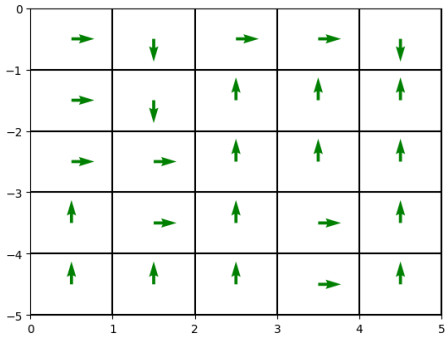

Figure 4: Policy to reach the state "Green"

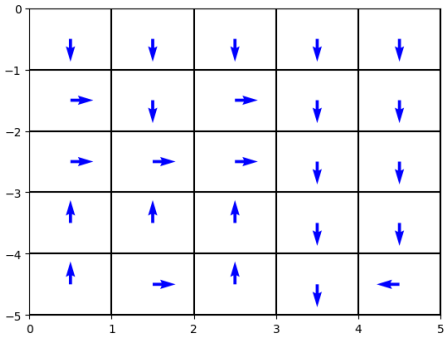

Figure 5: Policy to reach the state "Blue"

3 Methodology

Techniques Used: SMDP Q learning, Intra option Q learning, Options, Hyperparamneter tuning, Identifying options, Network theory

3.1 Policies for options

Every option is defined by the tuple I, π, β, where 'I' are the states where the option can be initiated, $'\pi'$ is the policy for the given option and β is the termination criteria for the given option.

The four options defined for Taxi-v3 environment are four options to reach the states "Yellow","Blue","Green" and "Red" respectively.

Because we have defined 4 options for the taxi to execute, it is important to define the initiation,policies and termination for the given options.

- Initiation: All four options can be initiated in all 500 states
- Termination: Option to reach Yellow terminates once the taxi reaches the position of Yellow, to reach green terminates after the taxi reaches Green, option to reach blue terminates after reaching the state Blue and option to reach red terminates after reaching the state Red.
- The polices for options have been illustrated in the Figures 1,2,3 and 4

4 SMDP Q learning with given Options

Listing 1: SMDP Q learning

```
#primitive actions = [0,1,2,3,4,5]
#options = [6,7,8,9] (yellow,green,red,blue)
for ep in range(int(NUM_EPS)):
      state = env.reset()
      done = False
      r = 0
      t = 0
      while not done:
          eps = max((0.99**ep)*0.9,EPS)
          # Choose action
          action = return_action(state,eps,Q_SMDP)

          # Checking if primitive action
          if action < 6:
              # Perform regular Q-Learning update for state-action pair
              next_state, reward, done,_ = env.step(action)
              #r+= gamma**(tmax-1) * reward
              r+=reward
              t +=1
              update = reward + gamma * max(Q_SMDP[next_state]) - ...
                  Q_SMDP[state][action]
              Q_SMDP[state][action] +=  alpha*(update)
              Update_Frequency_SMDP_q_learning[state][action] += 1
              state = next_state

          # Checking if action chosen is an option

          if action≥6:
            reward_bar = 0
            current_state = state
            optdone = False
            tau = 0
            done_ = False
            while (optdone==False):
              taxi_row,taxi_col,_,_  = env.decode(state)
              optaction = options[action][taxi_row][taxi_col]
              next_state,reward,done,_ = env.step(optaction)
              tau +=1
              t+=1
              #accumulate the discounted reward, when the option is executing
              reward_bar = reward_bar + (gamma**(tau-1))*reward
              state = next_state
              row_t_1,col_t_1,_,_ = env.decode(next_state)
              next_state_is_terminal = True if (5*(row_t_1) + col_t_1) == ...
                  termination_dict[action] else False
              if next_state_is_terminal:
                optdone = True
            #update the value function for the option and the frequency table
            update = reward_bar + (gamma**(tau) * max(Q_SMDP[state])) - ...
                Q_SMDP[current_state][action]
            Q_SMDP[current_state][action] +=  alpha * update
            r+=reward_bar
            Update_Frequency_SMDP_q_learning[current_state][action] += 1
```

Listing 2: Evaluating the policy learnt

```
#the learnt policies have been evaluated by caluclating the expected ...
    reward and time taken to drop the passenger taken over 1000 episodes
def average_episode_time(Q):
    steps = []
    trajectories_ep = []
    rewards = []
    for i in range(1000):
      state = env.reset()
      done = False
      t = 0
      r = 0
      trajectory = []
      while not done:
        action = np.argmax(Q[state])
        if action<6:
          next_state,reward,done,_ = env.step(action)
          r+=reward
          trajectory.append([state,action,next_state,reward,False])
          state = next_state
          t+=1
        if action≥6:
          optdone = False
          while not optdone:
            row_t,col_t,_,_ = env.decode(state)
            optaction = options[action][row_t][col_t]
            next_state,reward,done,_ = env.step(optaction)
            r+=reward
            trajectory.append([state,optaction,next_state,reward,True])
            t+=1
            state= next_state
            row_t_1,col_t_1,_,_ = env.decode(next_state)
            next_state_is_terminal = True if (5*(row_t_1) + col_t_1) == ...
                termination_dict[action] else False
            if next_state_is_terminal:
              optdone = True
            state = next_state
      steps.append(t)
      rewards.append(r)
      trajectories_ep.append(trajectory)
    return(np.array(steps).mean()),(np.array(rewards).mean()),steps,trajectories_ep
```

4.1 Results

The results of learnt policies of SMDP Q learning in Taxi environment with primitive actions north,south, west and east and options taking the taxi to Red, Green, Yellow and Blue have been discussed below. **"Weights and Biases"** has been used for hyperparameter tuning.Hyperparameters tuned are

- learning rate "alpha"
- epsilon
- Number of episodes
- Discounting factor gamma fixed to 0.9

Figure 5 illustrates the summary of hyperparameter tuning using weights and Biases.

Figure 6 shows the reward curve for the best configuration of hyperparamters using SMDP Q learning. Time taken to solve the environment is illustrated in Figure 7

Figure 8, Figure 9 respectively the heat map of state values and update frequency of Q table obtained using SMDP Q learning

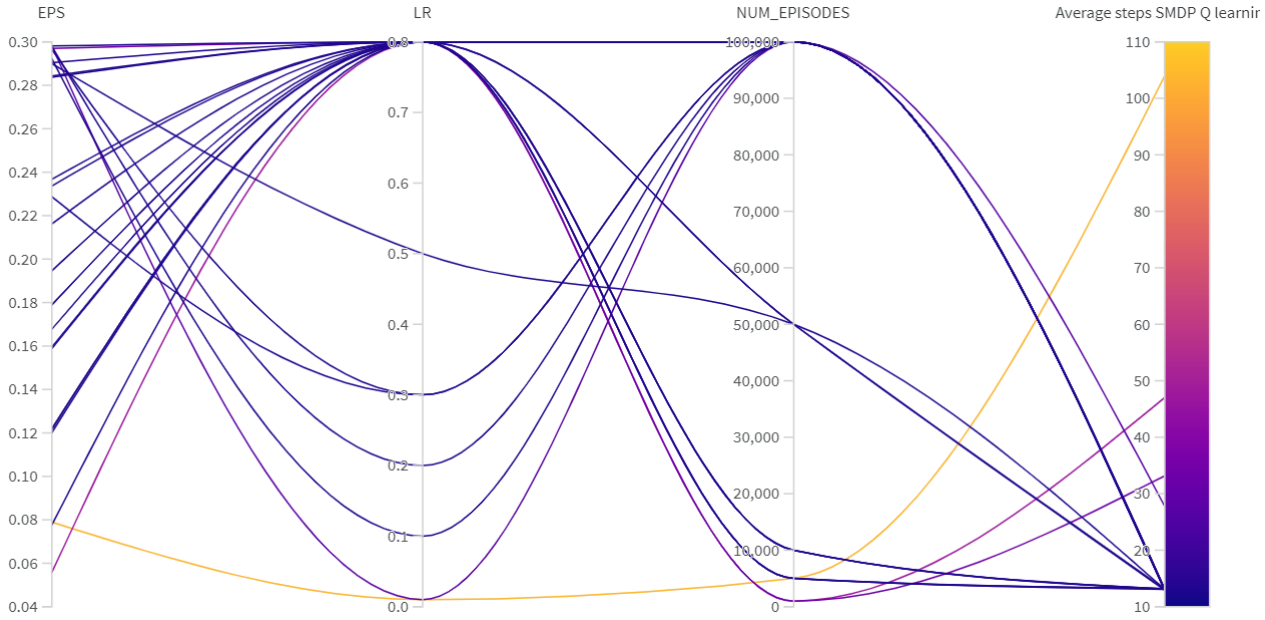

Figure 6: Summary of Hyperparameter tuning for SMDP Q learning

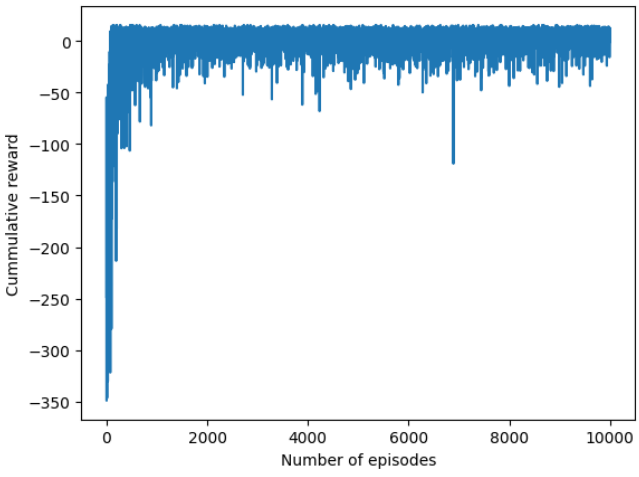

Figure 7: Reward curve for SMDP Q learning

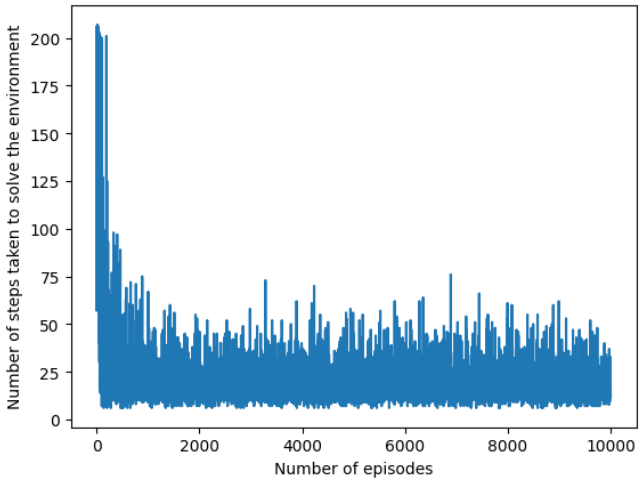

Figure 8: Number of episodes vs Time steps

Over an average of 1000 trails the avergae reward earned by the Taxi is 7.923 and average time taken to solve the episode is 13.077 steps.

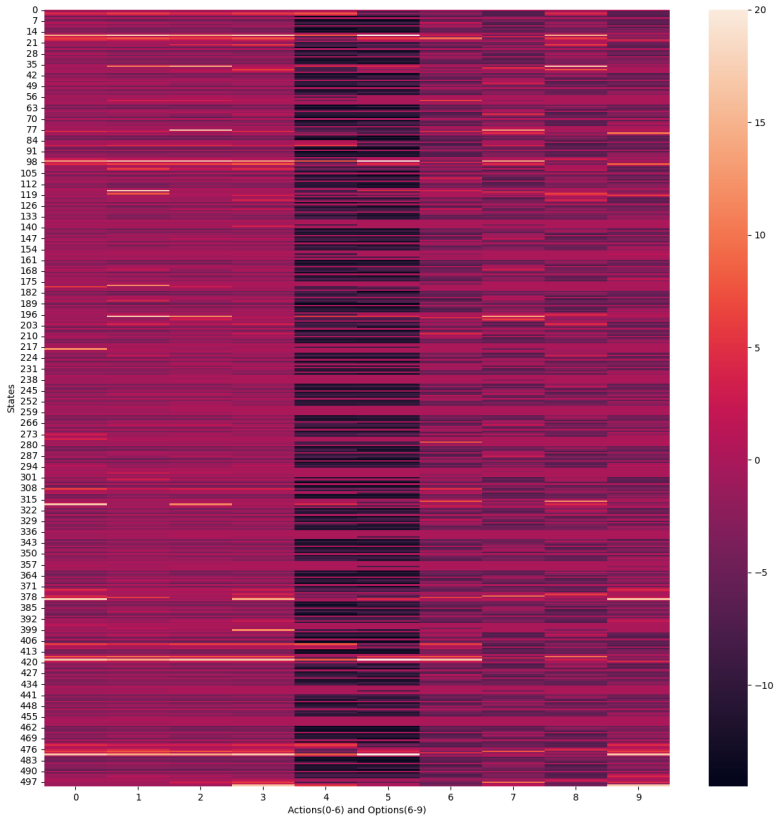

Figure 9: Heat map of Value function

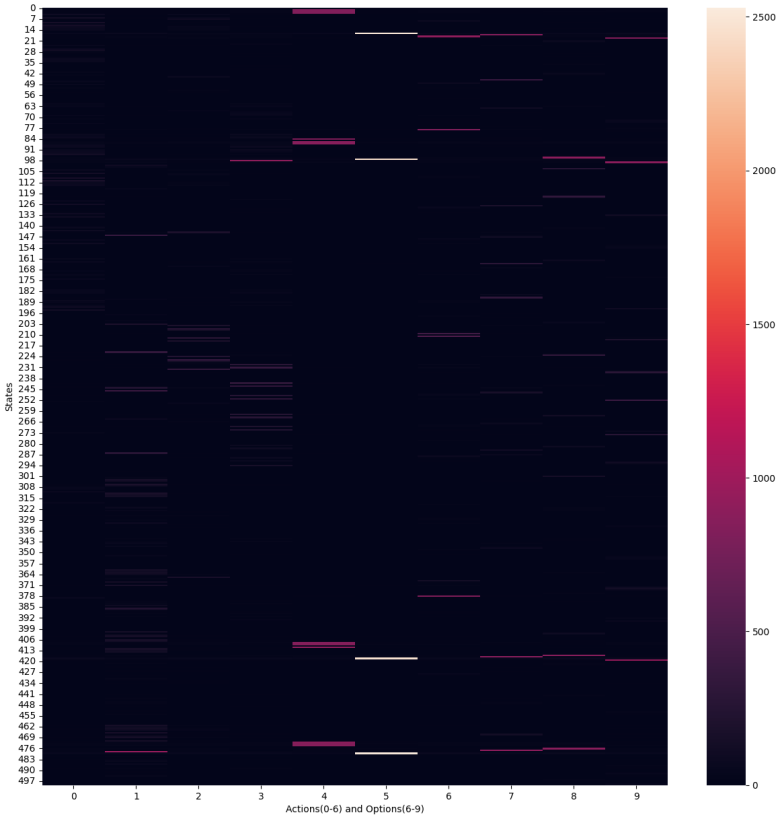

Figure 10: Update frequency table

5 Intra option Q learning with given Options

Listing 3: SMDP Q learning

```
def update_intraoption_action(state,action,next_state,reward): #here ... action is a primitive action
    Q_intra_option_Q[state][action] += alpha*(reward + ... gamma*max(Q_intra_option_Q[next_state]) - ... Q_intra_option_Q[state][action])
    Update_Frequency_intra_q_learning[state][action] +=1
    row_t,col_t,_,_ = env.decode(state)
    row_t_1,col_t_1,_,_ = env.decode(next_state)

    for option in range(6,10):
      optaction  = options[option][row_t][col_t]
      next_state_is_terminal = True if (5*(row_t_1) + col_t_1) == ... termination_dict[option] else False
      if optaction == action:
        if next_state_is_terminal == False:
          Q_intra_option_Q[state][option] += alpha*(reward + ... (gamma*Q_intra_option_Q[next_state][option]) - ... Q_intra_option_Q[state][action])
          Update_Frequency_intra_q_learning[state][option] += 1
        else:
          Q_intra_option_Q[state][option] += alpha*(reward + ... gamma*max(Q_intra_option_Q[next_state]) - ... Q_intra_option_Q[state][option])
          Update_Frequency_intra_q_learning[state][option] += 1

  for ep in range(int(NUM_EPS)):
    state = env.reset()
    done = False
    tmax = 0
    r = 0
    while not done:
      eps = max((0.99**ep)*0.9,EPS)
      action = return_action(state,eps,Q_intra_option_Q)
      if action <6:
        next_state,reward,done,_ = env.step(action)
        r+= gamma**(tmax-1) * reward
        tmax+=1
        update_intraoption_action(state,action,next_state,reward)
        state = next_state
        #print(state,action,next_state,reward)
      if action≥6:
        #print("current_option: ",action)
        optdone = False
        while not optdone:

          row_t,col_t,_,_ = env.decode(state)
          optaction = options[action][row_t][col_t]
          next_state,reward,done,_ = env.step(optaction)
          r+= gamma**(tmax-1) * reward
          tmax+=1
          update_intraoption_action(state,optaction,next_state,reward)
          state= next_state
          row_t_1,col_t_1,_,_ = env.decode(next_state)
          next_state_is_terminal = True if (5*(row_t_1) + col_t_1) == ... termination_dict[action] else False
          #print(state,optaction,next_state,reward,termination_dict[action])
          if next_state_is_terminal:
            optdone = True
    cumul_reward_intra.append(r)
```

5.1 Results

The results of learnt policies with Intra option Q learning in Taxi environment with primitive actions north,south, west and east and with options taking the taxi to Red, Green, Yellow and Blue have been discussed below. **"Weights and Biases"** has been used for hyperparameter tuning.
Figure 10 illustrates the summary of hyperparameter tuning using weights and Biases.
Figure 11 shows the reward curve for the best configuration of hyperparameters using Intra option Q learning. Time taken to solve the environment is illustrated in Figure 12
Figure 13, Figure 14 respectively the heat map of state values and update frequency of Q table obtained using Intra option Q learning

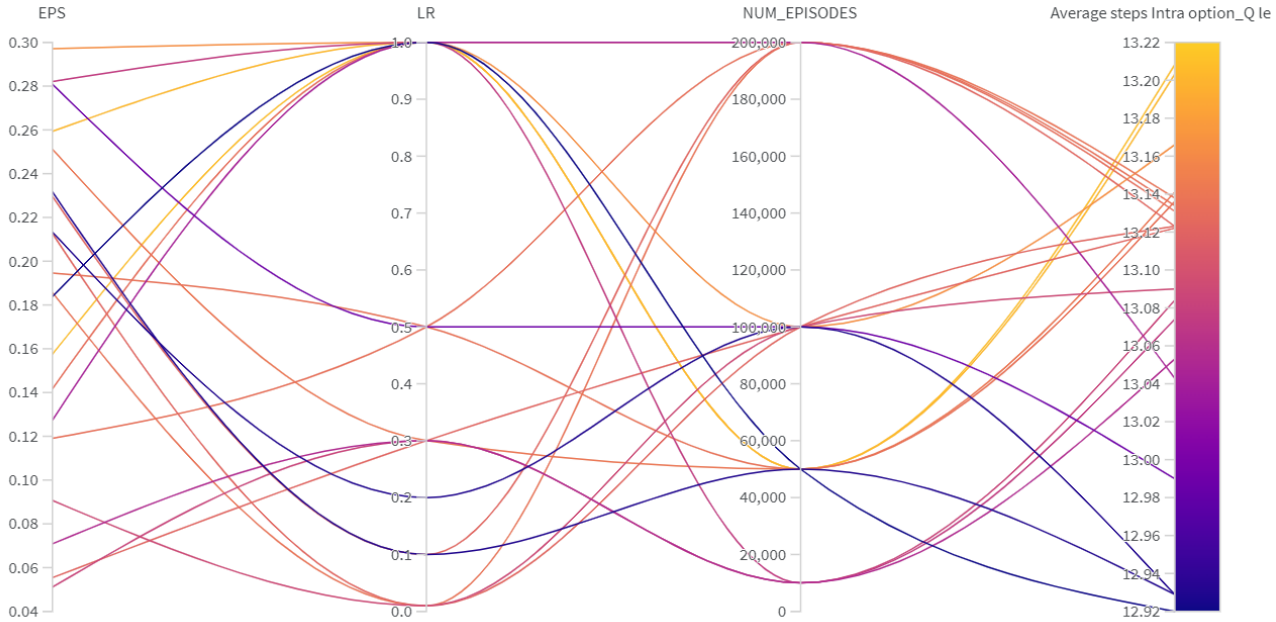

Figure 11: Summary of Hyperparameter tuning for Intra Option Q learning

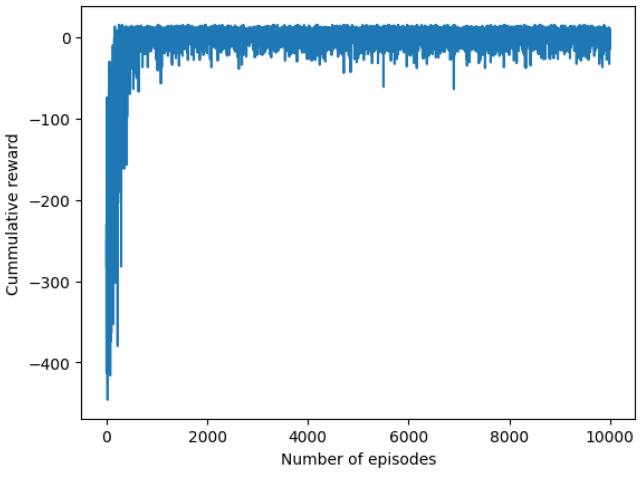

Figure 12: Reward curve for Intra Option Q learning

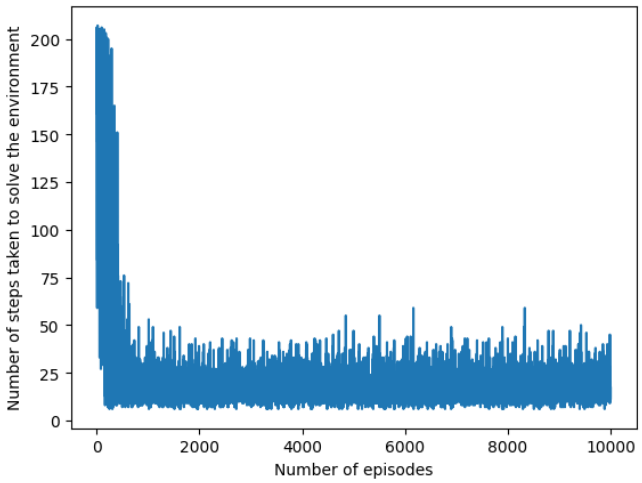

Figure 13: Number of episodes vs Time steps

Over an average of 1000 trails the average time taken to solve the episode is 13.053 steps. High variance solutions are neglected as they give some unstable policies. Thus, high variance solutions that give average time steps less than 13 approximately (12.9) have been neglected to output a stable policy.

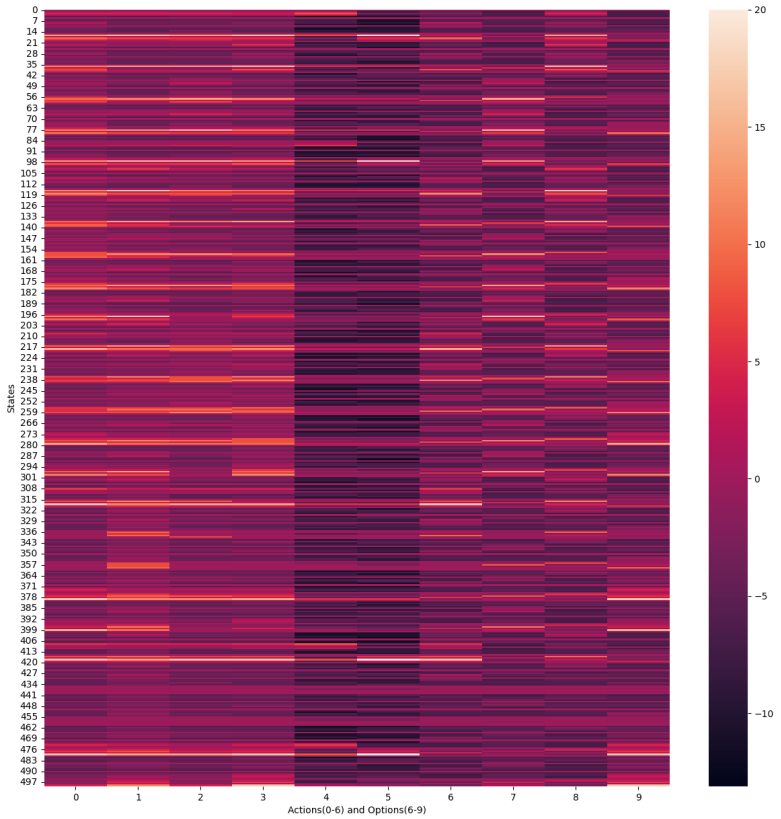

Figure 14: Heat map of Value function

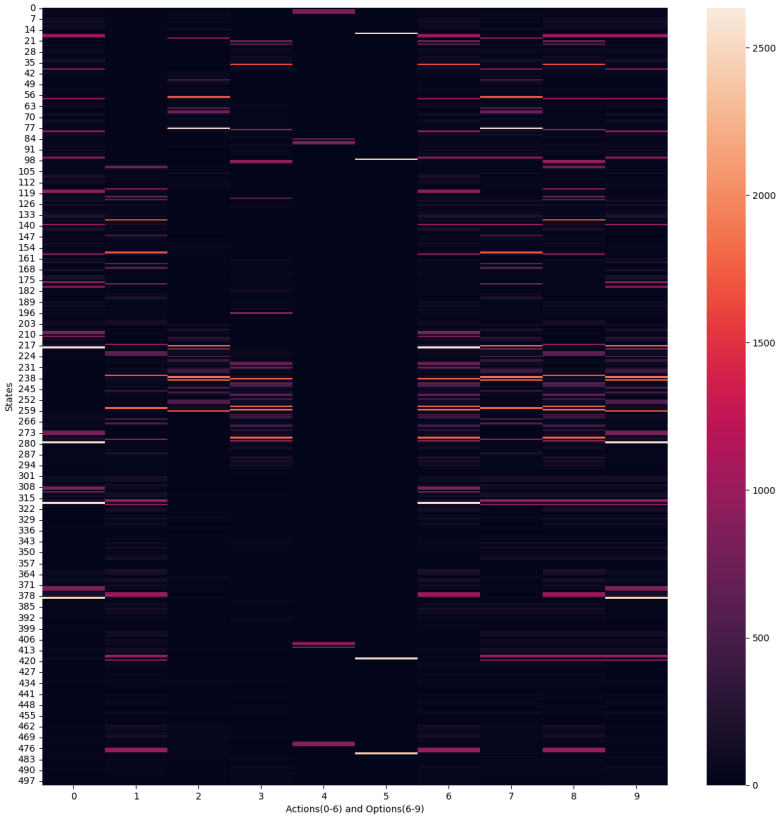

Figure 15: Update frequency table

6 Defining Mutually exclusive options

Now, we have the task of defining Mutually exclusive options with respect to given options.

6.1 Discovering options : Betweenness centrality

One of the popular ways of discovering useful options is to identify nodes with high betweenness centrality in the state space. The given environment MDP has been visualized as an un-directed graph, with edges representing the possible non-stochastic state transitions Figure 15 illustrates the "Taxi-v3" environment as a graph

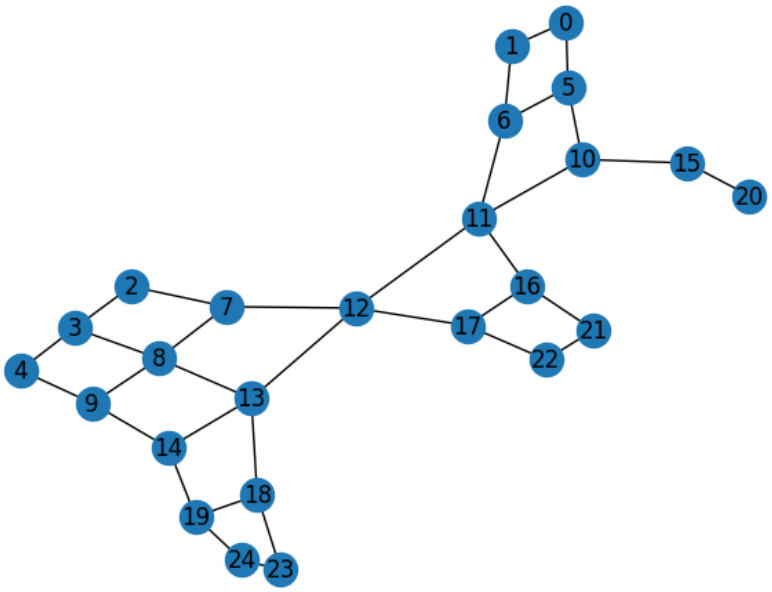

Figure 16: Visualizing Taxi-v3 environment as a graph

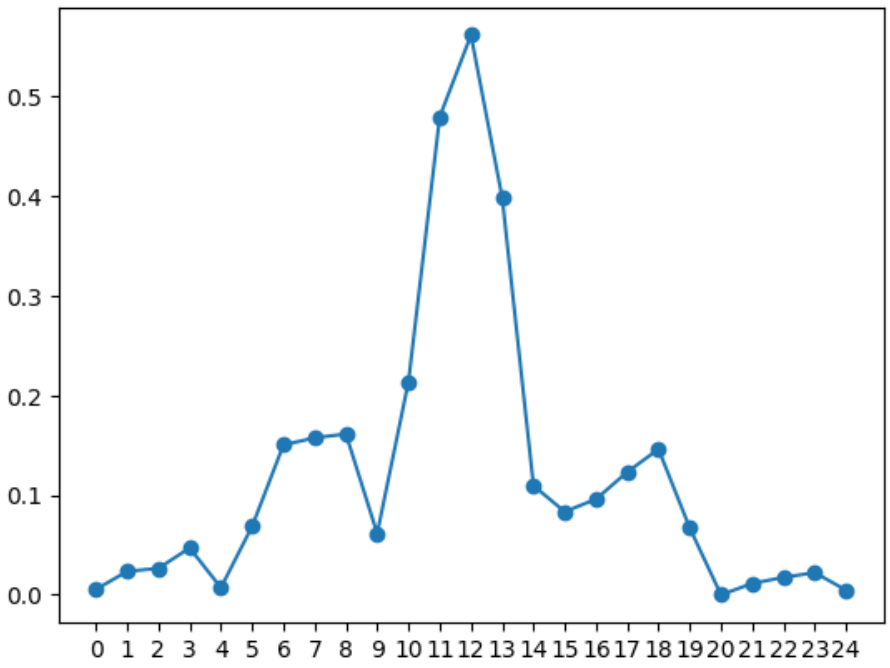

Figure 17: Betweenness centrality of nodes

As we can clearly infer from the graph, **states 11 and 12** have the top 2 highest betweenness centrality.

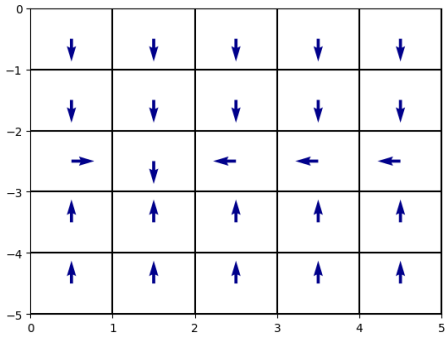

Figure 18: Policy to reach the state 11

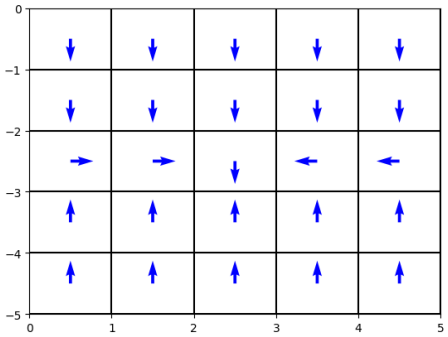

Figure 19: Policy to reach the state 12

Thus, two mutually exclusive options with respect to the initial options have been defined as to reach these two states 11 and 12.
As with any option, we need to define a tuple I, π, β for the two options defined above.

- Initiation: The option can be initiated in any state
- Termination: The option can be terminated, once the taxi reaches the location 11 for option taking the taxi to 11 and location 12 for option taking the taxi to 12
- Policy: The Figures 17 and 18 illustrate the policies of options to take the taxi to the states 11 and 12.

7 SMDP Q learning

7.1 Results

The results of learnt policies with SMDP Q learning in Taxi environment with primitive actions north,south, west and east and options taking the taxi to the states 11 and 12 have been discussed below. **"Weights and Biases"** has been used for hyperparameter tuning. Hyperparameters tuned are

- learning rate "alpha"
- epsilon
- Number of episodes
- Discounting factor gamma fixed to 0.9

Figure 19 illustrates the summary of hyperparameter tuning using weights and Biases.
Figure 20 shows the reward curve for the best configuration of hyperparamters using SMDP Q learning. Time taken to solve the environment is illustrated in Figure 21
Figure 22, Figure 23 respectively the heat map of state values and update frequency of Q table obtained using SMDP Q learning with these alternate set of options

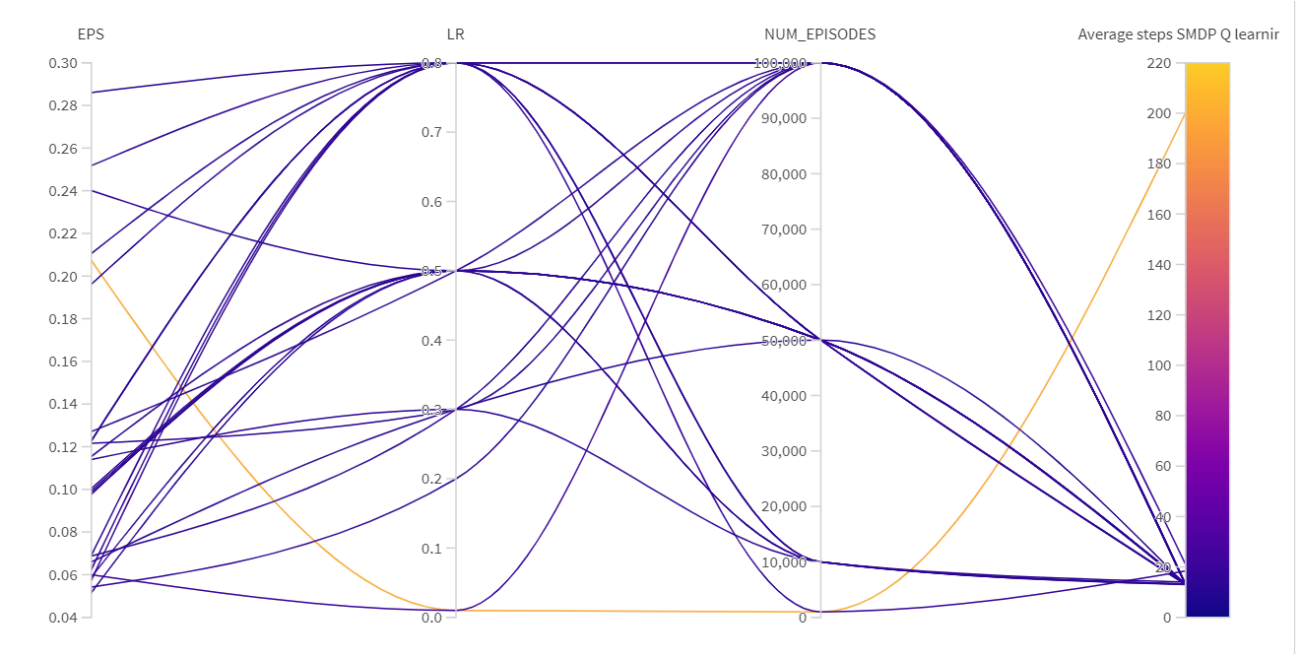

Figure 20: Summary of Hyperparameter tuning for SMDP Q learning for alternate set of options

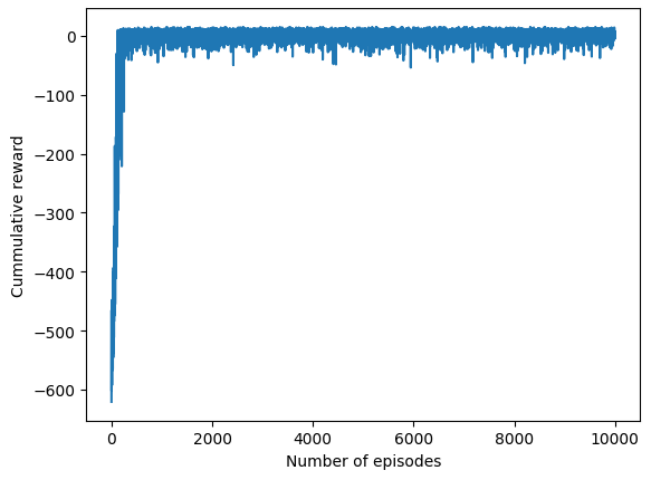

Figure 21: Reward curve for SMDP Q learning

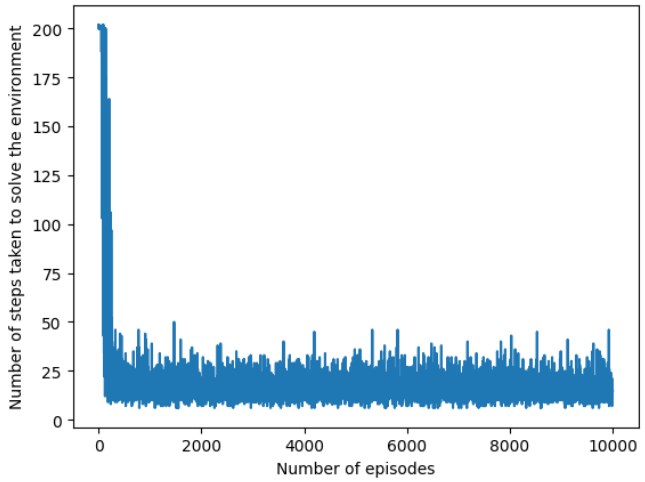

Figure 22: Number of episodes vs Time steps

Over an average of 1000 trails the average reward earned by the Taxi is 8.038 and average time taken to solve the episode is 12.964 steps, which is better than the previously defined options.

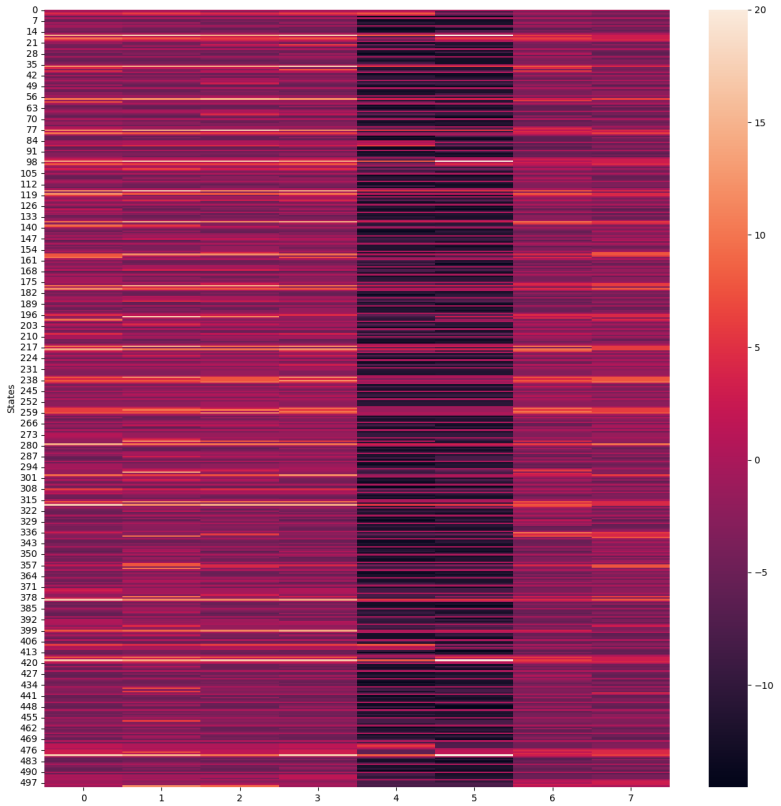

Figure 23: Heat map of Value function with Alternate options

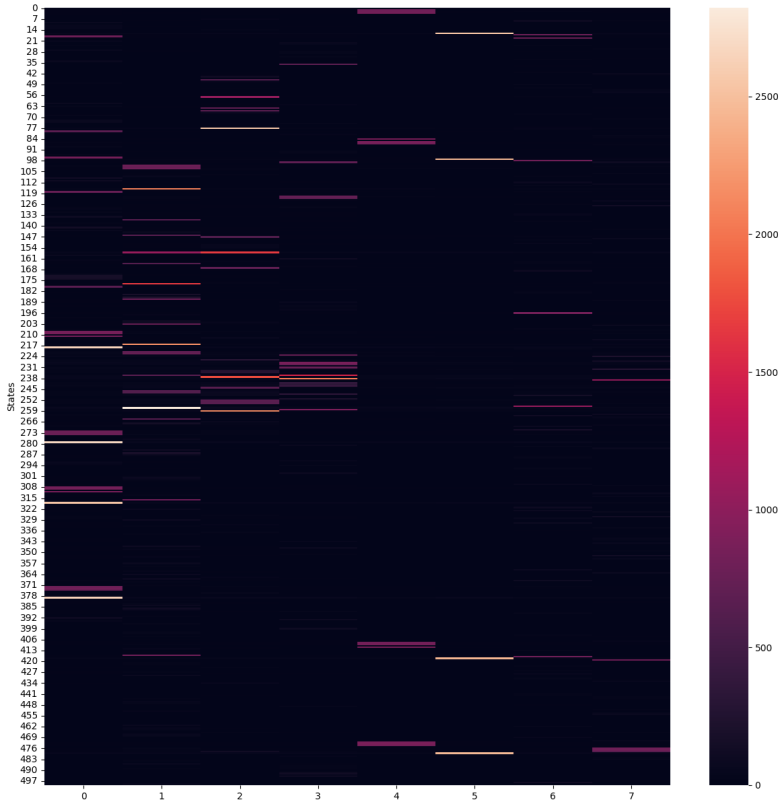

Figure 24: Update frequency table

8 Intra option Q learning

8.1 Results

The results of learnt policies with Intra option Q learning in Taxi environment with h, west and east and alternate options taking the taxi to states 11 and 12 have been discussed below. **"Weights and Biases"** has been used for hyperparameter tuning.
Figure 24 illustrates the summary of hyperparameter tuning using weights and Biases.
Figure 25 shows the reward curve for the best configuration of hyperparameters using Intra option Q learning. Time taken to solve the environment is illustrated in Figure 26
Figure 27, Figure 28 respectively the heat map of state values and update frequency of Q table obtained using Intra option Q learning

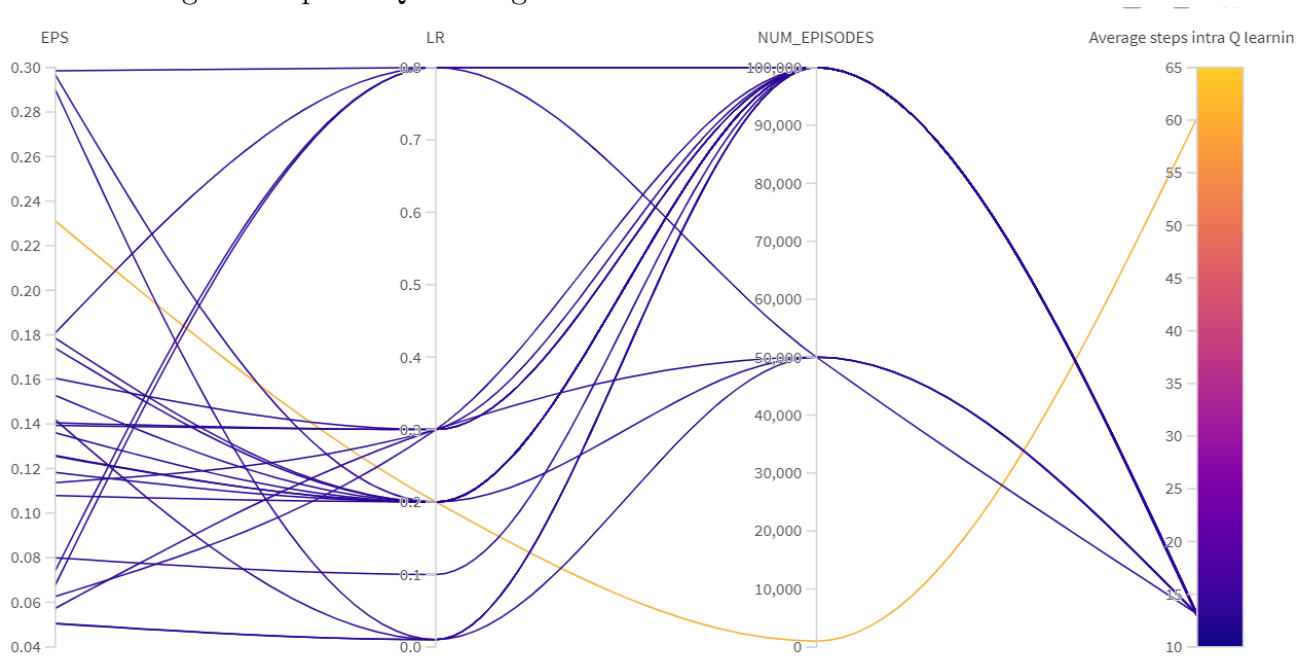

Figure 25: Summary of Hyperparameter tuning for Intra Option Q learning with Alternate options

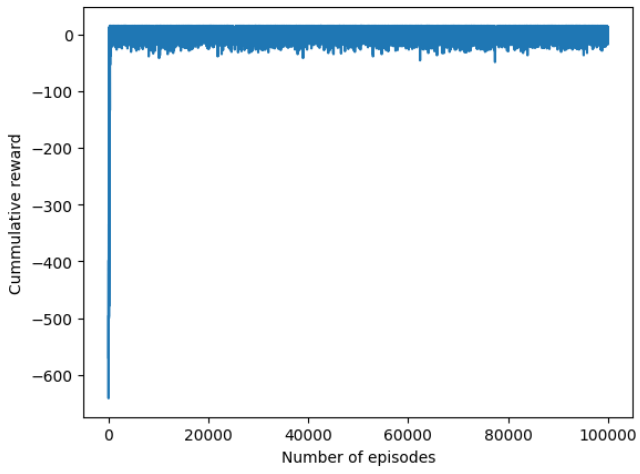

Figure 26: Reward curve for Intra Option Q learning

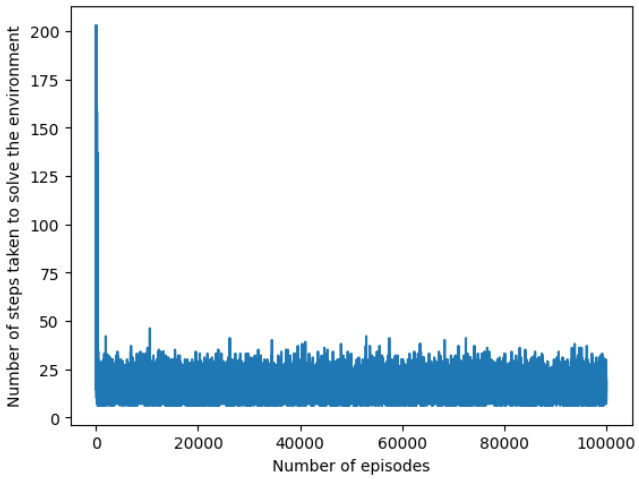

Figure 27: Number of episodes vs Time steps

Over an average of 1000 trails the average time taken to solve the episode is 13.018 steps which is better than the previous set of options taking them to Red, Green, Blue and Yellow

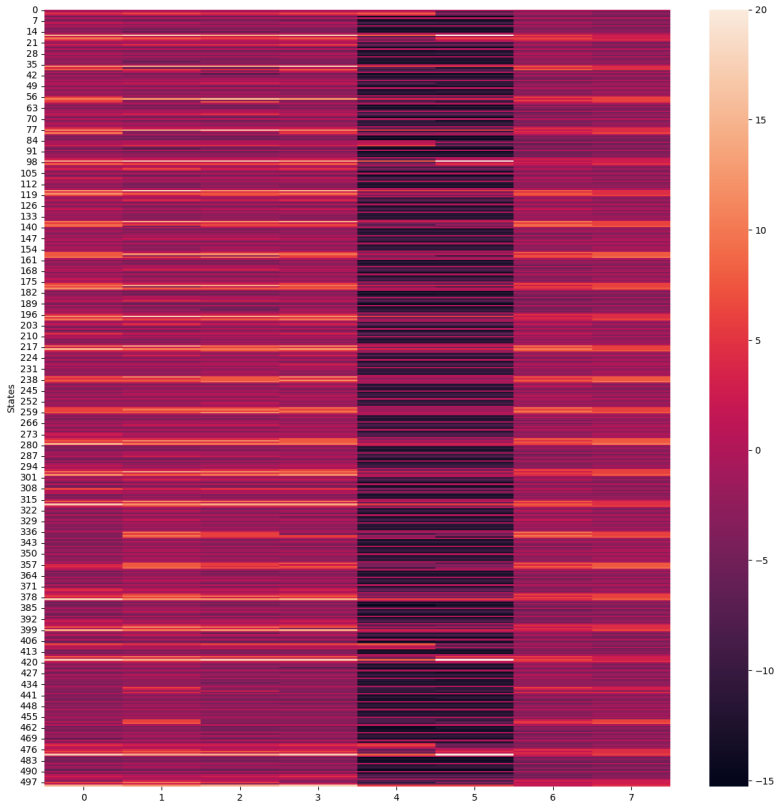

Figure 28: Heat map of Value function

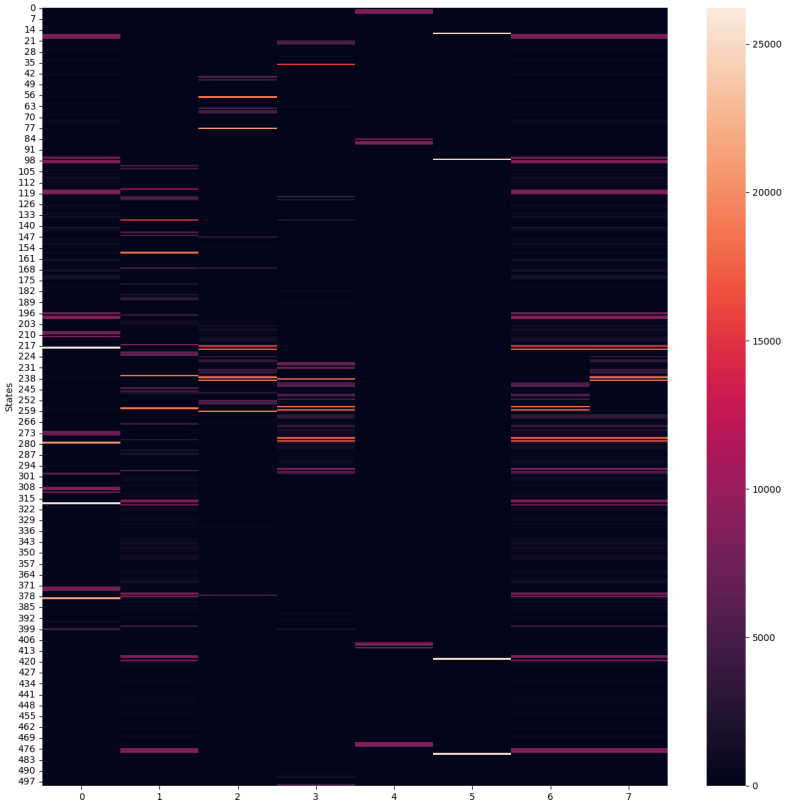

Figure 29: Update frequency table

9 Comparision of Intra Option Q learning vs SMDP Q learning

SMDP Q learning does not make efficient use of trajectories observed. It only updates the value function only after the termination of the option being executed. Thus, SMDP Q learning is low and makes fewer updates per episode.
The above limitations are overcame by Intra Option Q learning. Intra option Q learning makes efficient use of sample trajectories. For a given transition s, a, s', it not only updates the value function for the given action, but also updates the value function for all the trajectories that are consistent with the observed transition. Moreover, Intra option Q learning does not wait for the option to terminate. Instead, it updates the value function during the execution of option as well.

9.1 Comparison for given set of options

For the given environment of "Taxi-v3", the intra option Q learning performed slightly better compared to SMDP Q learning by solving the environment in just 12.9 steps. When we compare the stability of learnt policies, the variance in the reward curves and plot for time taken to solve the environment, the variance is lower if the learning algorithm used is Intra Option Q learning. Also ,when we observe the frequency of updates of intra option Q learning and SMDP Q learning, Intra Option Q learning has much higher number of updates compared to Intra option Q learning.

9.2 Comparison for alternate set of options

The performance of Intra option Q learning is more or less comparable to the performance of SMDP Q learning(slightly higher). There might be a couple of reasons on why the performance of both SMDP Q learning and Intra option Q learning are comparable. One of the reasons might be that the number of states is very small compared to the number of training episodes. Thus, both SMDP Q learning and Intra option Q learning might have already converged to their optimal values and thus their performance is comparable. Again, Intra Option Q learning showed very low variance even if the training is continued for huge number of episodes. From the Update frequency tables we can infer that the frequency of updates in Intra option Q learning is more than in SMDP Q learning.

10 Interpretation of learnt policies

10.1 Optimal policies learnt with given set of options

Before going to the explanation of learnt policies it is important to understand the notation used. Please note that action 6 corresponds to option that takes the taxi to state Yellow, similarly 7 corresponds to Green, 8 corresponds to Red and finally 9 corresponds to the option that takes the taxi to the Blue
1. Policies learnt with respect to passenger location
From the Figures 29,30,31,32, it is evident that, whenever the passenger location is at yellow, the taxi is choosing the corresponding option 6 considerable amount of times. Similarly if the passenger is at Green, the corresponding option chosen is 7, if passenger is at Red, the corresponding option chosen is 8 and finally when the passenger is at location Blue, the corresponding option chosen is 9.
2 . Policies learnt with respect to Destination location

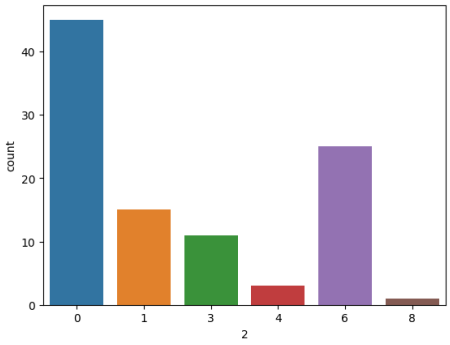

Figure 30: Passenger at state "Yellow"

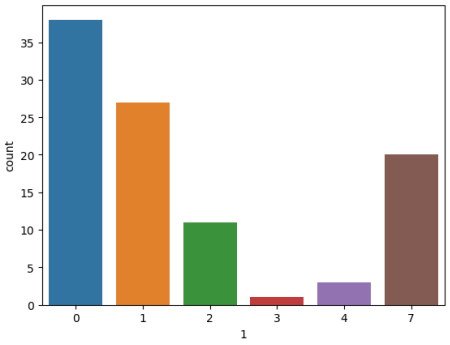

Figure 31: Passenger at state "Green"

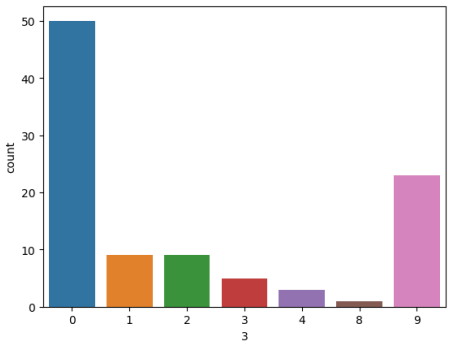

Figure 32: Passenger at state "Blue"

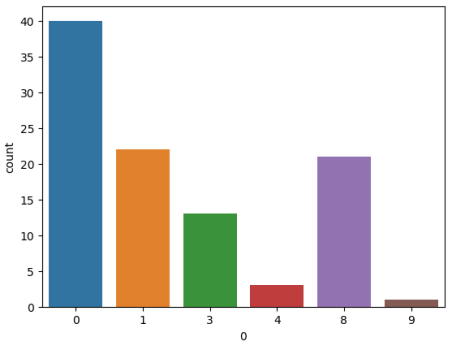

Figure 33: Passenger at state "Red"

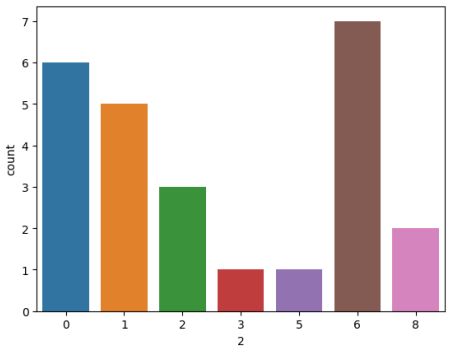

Figure 34: Passenger at state "Yellow"

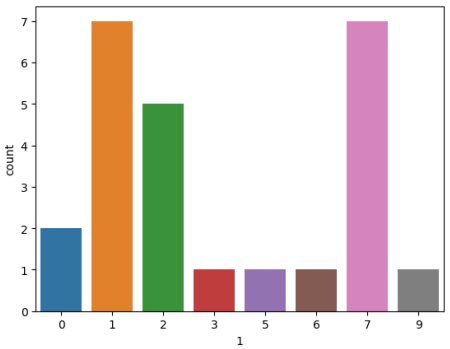

Figure 35: Passenger at state "Green"

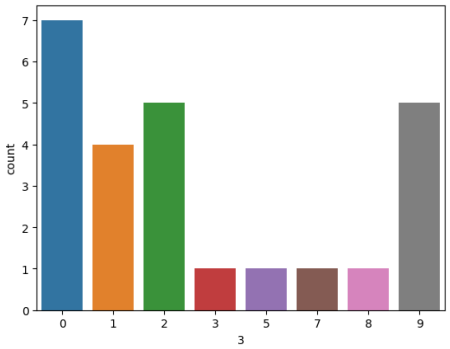

Figure 36: Passenger at state "Blue"

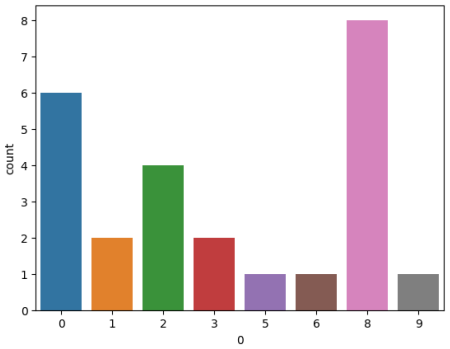

Figure 37: Passenger at state "Red"

From the Figures 33,34,35,36, it is evident that, whenever the destination location is at yellow, the taxi is choosing the corresponding option 6 considerable amount of times. Similarly if the destination is at Green, the corresponding option chosen is 7, if destination is at Red, the corresponding option chosen is 8 and finally when the destination is at location Blue, the corresponding option chosen is 9.
Thus, apart from primitive actions, the policy learnt by the Taxi agent is consistent with our intuition and we can say that the taxi has learnt a reasonable policy.
In a nutshell the learnt policy can be defined as:
1.Observe passenger location/ Destination
2.Perform the option that takes you to passenger location/ Destination

10.2 Optimal policies learnt with Alternate set of options

Before going to the explanation of learnt policies it is important to understand the notation used. Please note that action 6 corresponds to option that takes the taxi to state 11, similarly 7 corresponds to the option that takes it to state 12

1. Policies learnt with respect to passenger location

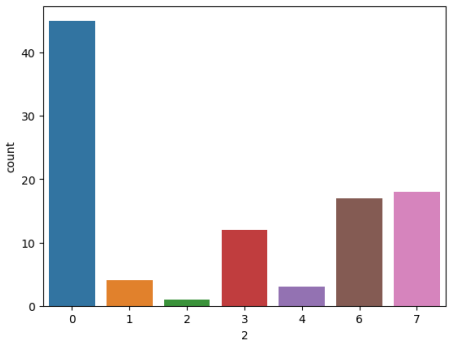

Figure 38: Passenger at state "Yellow"

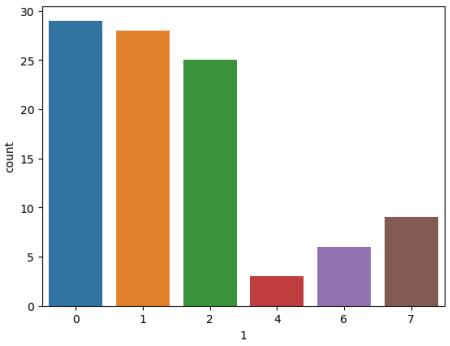

Figure 39: Passenger at state "Green"

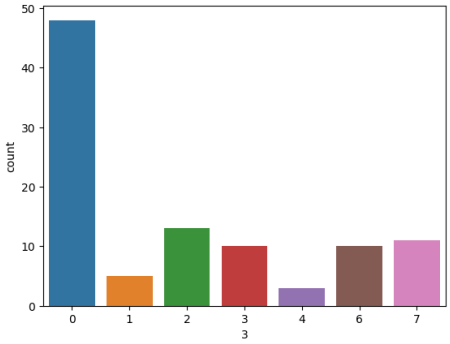

Figure 40: Passenger at state "Blue"

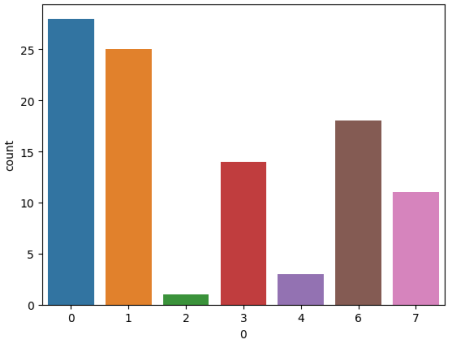

Figure 41: Passenger at state "Red"

To reach the state "Yellow" and "Red" the taxi reached state 11 considerable amount of times when compared to state 12(Although for yellow this wasn't very significant). Thus, the taxi has learnt a policy that chooses the state that is nearest to passenger location that is reachable via available options.

2 . Policies learnt with respect to Destination location

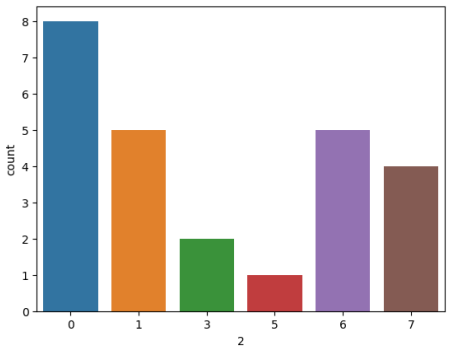

Figure 42: Passenger at state "Yellow"

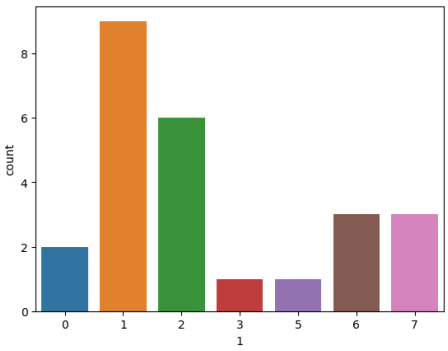

Figure 43: Passenger at state "Green"

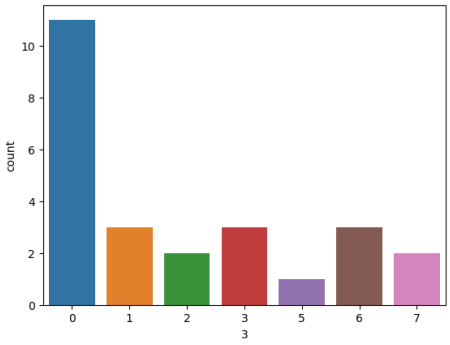

Figure 44: Passenger at state "Blue"

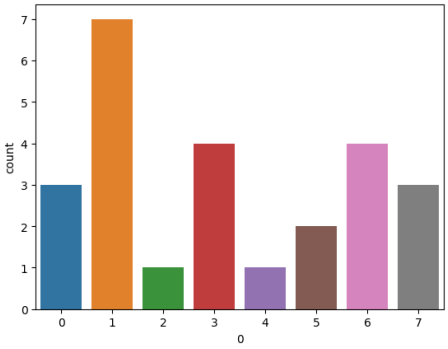

Figure 45: Passenger at state "Red"

Primitive options are fine to reach open states such as Green and Blue. Thus, the agent will do just fine even with primitive actions. But to reach corner states like Red and Yellow, is it is useful for the agent to make use of available options and then perform primitive options. When we observe the Figures 41,42,43 and 44, the agent exactly follows out intuition. It is preferring options 6 and 7, when the destination is at Red and Yellow and is preferring both primitive actions and options to reach the states blue and Green.
In a nutshell the learnt policy can be defined as:
1.Observe passenger location/ Destination
2.Reach middle of state space (states 11 and 12)
3.Decide on primitive actions that take you to the desired location

11 Conclusion

1. SMDP Q learning and Intra Option Q learning are implemented on Taxi-v3 environment
2. Both set of options performed equivalently well on the Taxi environment as the state space is small compared to episodes trained. The alternate set of policies identified by using betweenness centrality of nodes performed slightly better compared to given options to move the taxi to each of the four designated locations namely, Yellow, Blue, Red, Green.
3. The taxi learnt reasonable policies consistent with our intuition as stated in Section 10.
4. Although not analyzed in this report, another three mutually exclusive options have been defined and worked on the Taxi environment. Here three options are to move taxi to 11,12 and 13 states instead of only 11 and 12. The notebook for the same have been shared in the submitted folder.
5. **Intra Option Q learning performs better than SMDP Q learning**

———————————-THE END———————————-