Programming Assignment - 3

Kamalesh Kumar K ce20b054
Kamalesh M ce20b055

April 2023

1 Introduction

In this assignment, we will solve the 'Taxi-v3' environment from OpenAI gym. SMDP and Intra option Q-Learning algorithms are implemented using the options framework in solving the environment. Finally, we compare the efficiency of both the agents in solving the environment.

2 SMDP Q-Learning for the given options

2.1 Defining the options

Four different options are used to solve the environment with SMDP Q-Learning. The options used are to reach the pickup or drop location defined by the environment as R,B,G,Y. Once the taxi reaches the required location, the drop/pick up primitive actions are deterministically chosen. Otherwise, the four primitive action: North, South, East, West are chosen using an ϵ-greedy policy defined separately for each option. The code for each option is listed below:

1. Reach R and pickup or drop:

```python

def Option_0(env,state,Q,eps=0.1):
  optdone = False
  x,y,pas,drop=env.decode(state)

  if (x==goal[0][0] and y==goal[0][1]):
      optdone = True
      if pas == 0:
        optact = 4
      elif drop == 0:
        optact = 5
      else:
        optact = 1
  else:
    optact = egreedy_policy(Q[0], 5*x+y, epsilon=eps)
  return [optact,optdone]
```

Listing 1: Option for reaching R for pickup or drop

2. Reach G and pickup or drop:

```python

def Option_1(env,state,Q,eps=0.1):
  optdone = False
  x,y,pas,drop=env.decode(state)

  if (x==goal[1][0] and y==goal[1][1]):
      optdone = True
      if pas == 1:
        optact = 4
      elif drop == 1:
        optact = 5
      else:
        optact = 1
  else:
    optact = egreedy_policy(Q[1], 5*x+y, epsilon=eps)
  return [optact,optdone]
```

Listing 2: Option for reaching G for pick up or drop

3. Reach B and pickup or drop:

```python

def Option_2(env,state,Q,eps=0.1):
  optdone = False
  x,y,pas,drop=env.decode(state)

  if (x==goal[2][0] and y==goal[2][1]):
      optdone = True
      if pas == 2:
        optact = 4
      elif drop == 2:
        optact = 5
      else:
        optact = 0
  else:
    optact = egreedy_policy(Q[2], 5*x+y, epsilon=eps)
  return [optact,optdone]
```

Listing 3: Option for reaching B for pickup or drop

4. Reach Y and pickup or drop:

```python

def Option_3(env,state,Q,eps=0.1):
  optdone = False
  x,y,pas,drop=env.decode(state)

  if (x==goal[3][0] and y==goal[3][1]):
      optdone = True
      if pas == 3:
        optact = 4
      elif drop == 3:
        optact = 5
      else:
        optact = 0
  else:
    optact = egreedy_policy(Q[3], 5*x+y, epsilon=eps)
  return [optact,optdone]
```

Listing 4: Option for reaching Y for pickup or drop

2.2 Computing $r(s, o)$

The discounted rewards for each option is computed as below:

```python
reward_bar = gamma*reward_bar + reward
```

2.3 Updating $Q(s_t, o)$

The option-value function are updated as per Q-learning as expressed below:

```
q_values_SMDP[subPrev, option] += alpha*(reward_bar + (gamma**move)*np.max(
    q_values_SMDP[subState, :]) - q_values_SMDP[subPrev, option])
```

2.4 Plots

The plot of rewards and steps taken as a function of episodes is shown in the graphs below

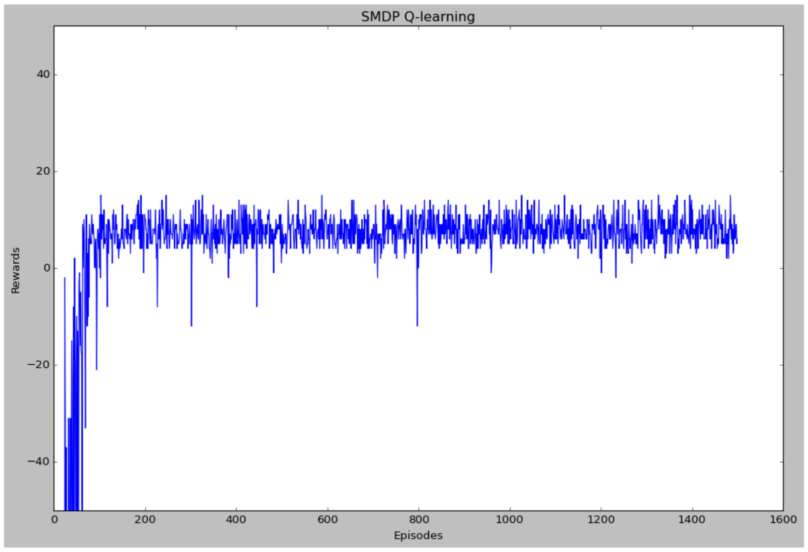

Figure 1: Plot of rewards for SMDP Q-Learning for the given set of options.

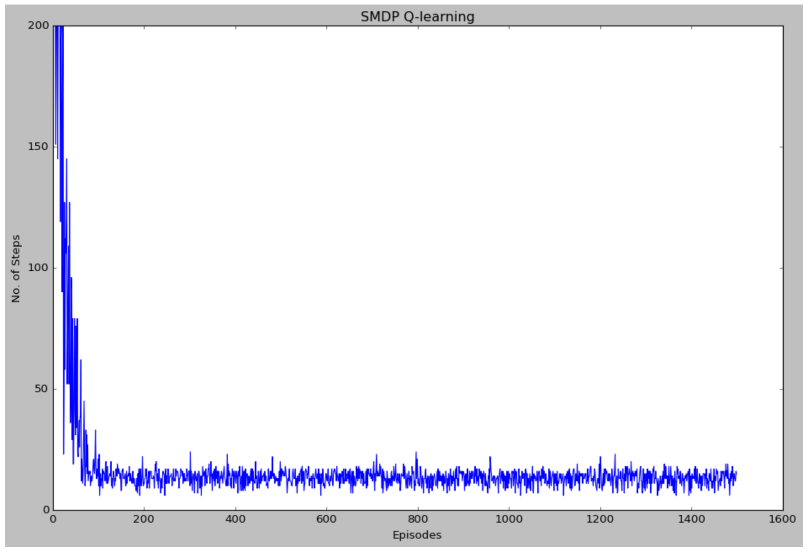

Figure 2: Plot of steps for SMDP Q-Learning for the given set of options.

For learning a policy over options, every possible passenger state, along with the possible drop locations, is considered as a state. If the passenger is in the one of the locations, the state will defined as passenger location and drop location with which the agent chooses the option which goes to the pickup location and picks up the passenger and once the passenger is picked up the passenger location changes to 4(inside the taxi) which changes the state and new option is chosen with respect to that state to reach the drop location. The plots of the reward and steps above clearly show convergence to an optimal option policy.

The policy learnt through SMDP Q-Learning for each of the four options can be visualized in the heatmap shown below:

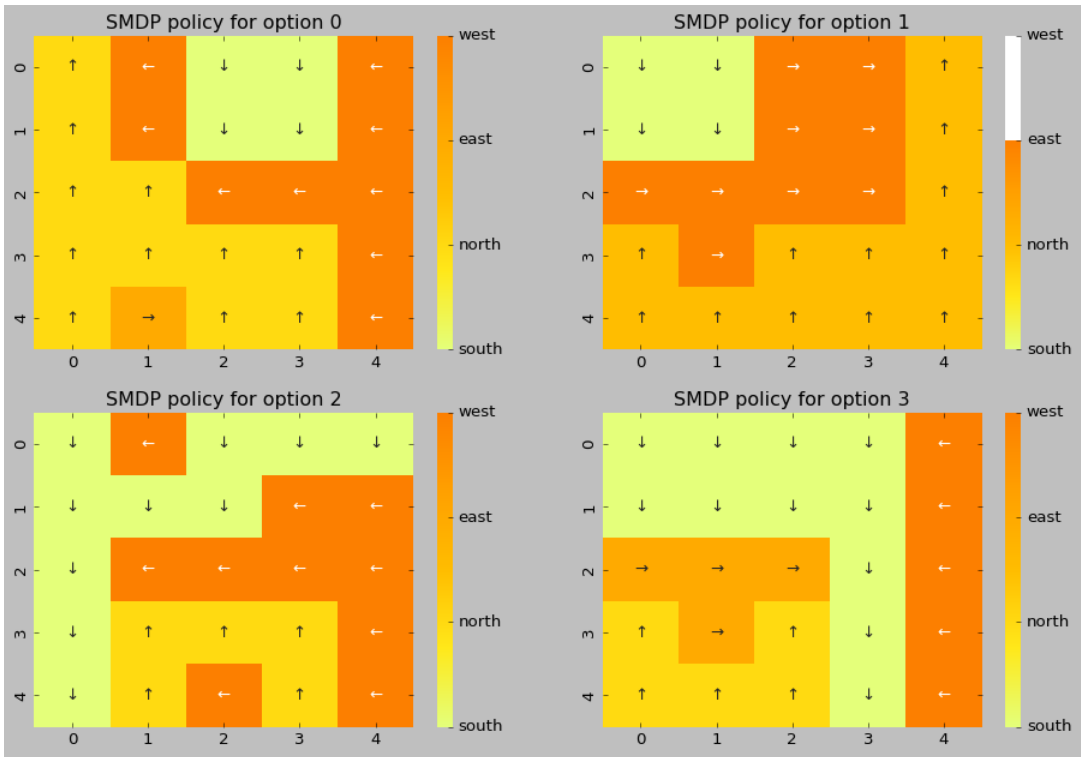

Figure 3: Visualization of the learned policy for each option for SMDP Q-Learning

On visualizing the learned policy above, we can see that the options defined have learned the right primitive action for each location in the 5×5 taxi grid-world. Even though in this setting, we don't consider the policy within the option to be fixed/given, it is also learned along with the policy over the options. The SMDP Q-Learning framework determines the frequency in which any option is selected (based on an ϵ-greedy policy). Thus consequently, the policy within the option is also learned(also using an ϵ-greedy policy) and updated in synchronization with the SMDP policy.

3 Intra option Q-Learning (IOQL) for the given options

For the IOQL setting, the given options are defined in a similar fashion to the SMDP setting, which was detailed in the previous section.

3.1 Updating $Q(s_t, o)$

The reward computation is done in a one-step fashion, and the following update rule for the Q-values is incorporated:

```

if optact_o == optact:
    if optdone_o:
        q_values_intra_option[Sub(state), o] += alpha*(reward + gamma*np.max(
    q_values_intra_option[Sub(next_state), :]) - q_values_intra_option[Sub(state), o])
    else:
        q_values_intra_option[Sub(state), o] += alpha*(reward + gamma*
    q_values_intra_option[Sub(next_state), o] - q_values_intra_option[Sub(state), o])
```

Listing 5: Updating Q-values for IOQL

As per the IOQL framework, the action values for all the primitive action within an option are also updated along with the option values for the states that occurred in the trajectory of the selected option. All other options in the option space that are concurrent with the selected option for a primitive action, are also updated. When an option is not done, the TD-update equation for option vales is similar to SARSA, whereas it is Q-Learning when an option terminates. The same is also reflected in the if-else condition above.

3.2 Plots

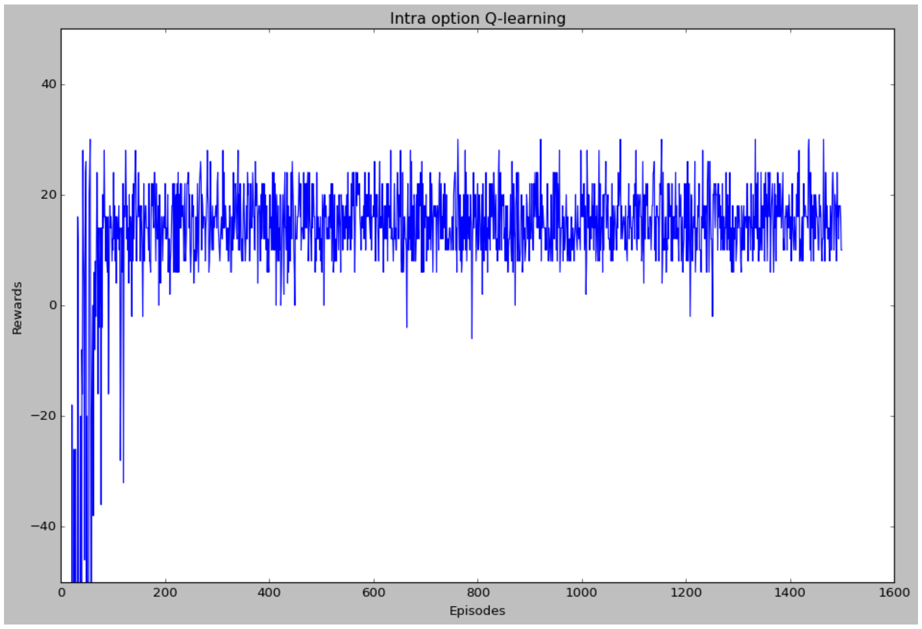

Figure 4: Plot of rewards for IOQL for the given set of options

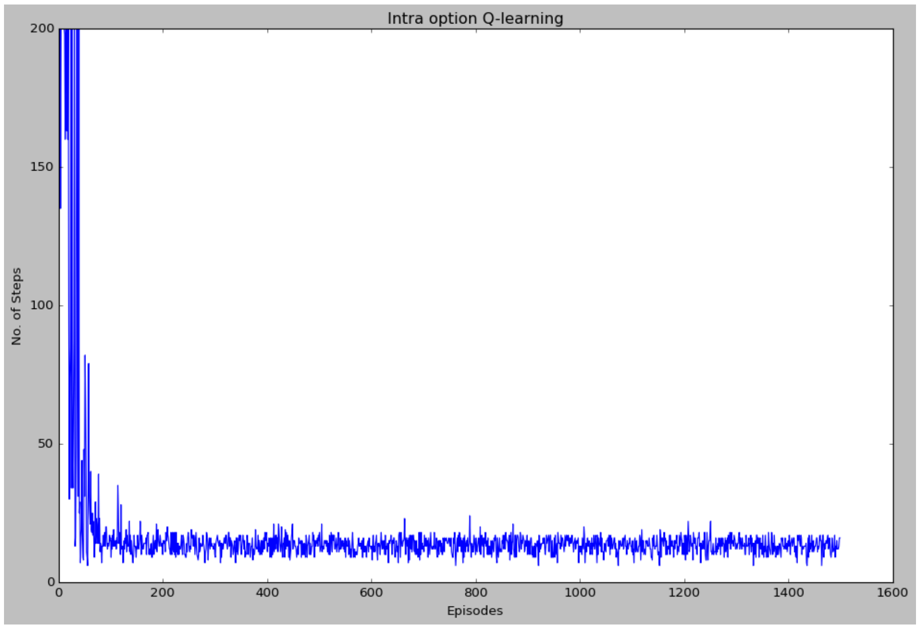

Figure 5: Plot of steps taken for IOQL for the given set of options

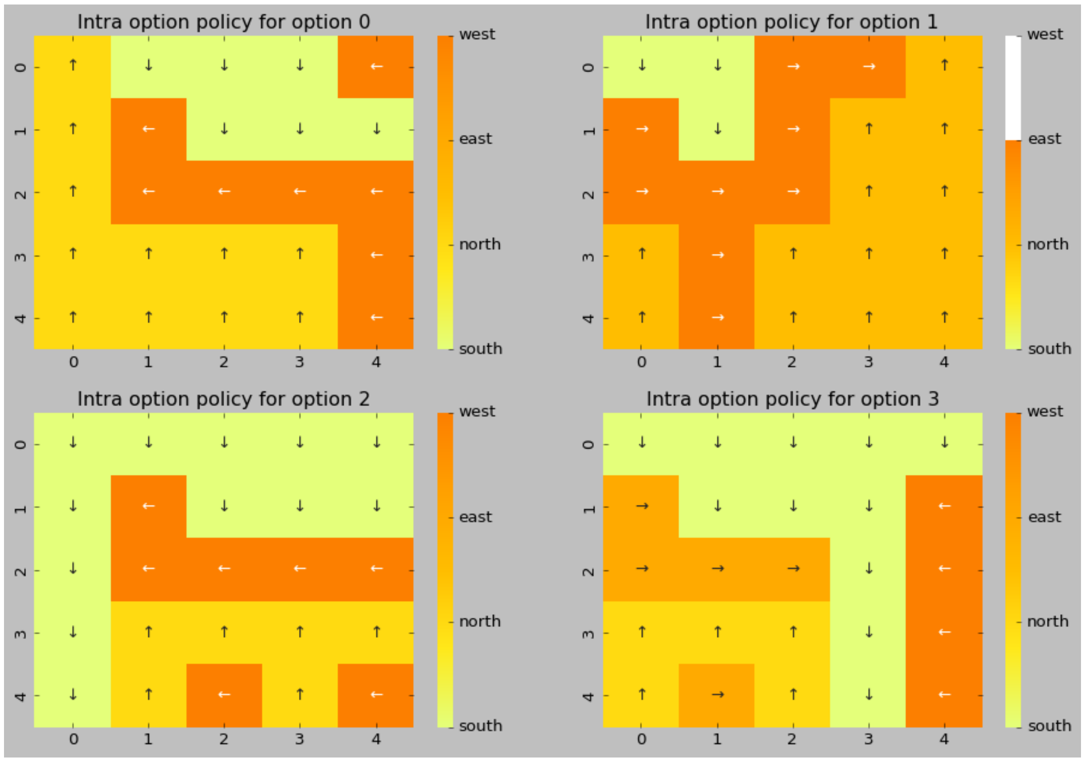

Figure 6: Visualization of the learned policy for each option for IOQL Q-Learning

4 SMDP and Intra option Q-Learning for alternate options

4.1 Defining the options

For alternate options to solve the taxi environment, we consider a total of six options. The first pair of options is for reaching the squares on the outer edges present on the horizontal center line. For each of these options, there exist two other options for reaching the two nearest goal states (for pick up or drop). It is important to note that the last four options are possible to be taken only when the taxi has reached the outer edge square using the first two options. Once it has reached one of these two squares, the possible options for the other square cannot be taken on the current square and vice-versa. The options to reach these outer-edge squares act as bottleneck options, as in many cases, they have to be crossed to reach the goal locations.

The code snippet for the new set of options is as below:

```python

nO = 6 #number of options
goal = [[0,0],[0,4],[4,0],[4,3],[2,1],[2,3]]

def Option_0(env,state,Q,eps=0.1):
  optdone = False
  x,y,pas,drop=env.decode(state)

  if (x==goal[0][0] and y==goal[0][1]):
      optdone = True
      if pas == 0:
        optact = 4
      elif drop == 0:
        optact = 5
      else:
        optact = 1
  else:
    optact = egreedy_policy(Q[0], 5*x+y, epsilon=eps)
  return [optact,optdone]

def Option_1(env,state,Q,eps=0.1):
  optdone = False
  x,y,pas,drop=env.decode(state)

  if (x==goal[1][0] and y==goal[1][1]):
      optdone = True
      if pas == 1:
        optact = 4
      elif drop == 1:
        optact = 5
      else:
        optact = 1
  else:
    optact = egreedy_policy(Q[1], 5*x+y, epsilon=eps)
  return [optact,optdone]

def Option_2(env,state,Q,eps=0.1):
  optdone = False
  x,y,pas,drop=env.decode(state)

  if (x==goal[2][0] and y==goal[2][1]):
      optdone = True
      if pas == 2:
        optact = 4
      elif drop == 2:
        optact = 5
      else:
        optact = 0
  else:
    optact = egreedy_policy(Q[2], 5*x+y, epsilon=eps)
  return [optact,optdone]

def Option_3(env,state,Q,eps=0.1):
  optdone = False
  x,y,pas,drop=env.decode(state)

  if (x==goal[3][0] and y==goal[3][1]):
      optdone = True
      if pas == 3:
        optact = 4
      elif drop == 3:
        optact = 5
      else:
        optact = 0
  else:
    optact = egreedy_policy(Q[3], 5*x+y, epsilon=eps)
  return [optact,optdone]

def Option_4(env,state,Q,eps=0.1):
  optdone = False
  x,y,pas,drop=env.decode(state)
  env2=copy.deepcopy(env)
  optact = egreedy_policy(Q[4], 5*x+y, epsilon=eps)

  next_state,_,_,_=env2.step(optact)
  x1,y1,_,_=env2.decode(next_state)
  if (x1==goal[4][0] and y1==goal[4][1]):

    optdone=True

  return [optact,optdone]

def Option_5(env,state,Q,eps=0.1):
  optdone = False
  x,y,pas,drop=env.decode(state)
  env2=copy.deepcopy(env)
  optact = egreedy_policy(Q[5], 5*x+y, epsilon=eps)

  next_state,_,_,_=env2.step(optact)
  x1,y1,_,_=env2.decode(next_state)
  if (x1==goal[5][0] and y1==goal[5][1]):

    optdone=True

  return [optact,optdone]
```

Listing 6: The alternate set of options

The Option4 and Option5 used above the are one used to reach the middle left and the middle right of the grid world.

The reward and Q-value updation function used are same as the one used section 2 and 3.

4.2 Plots for SMDP Q-Learning

The plot of rewards and steps taken as a function of episodes is shown in the graphs below

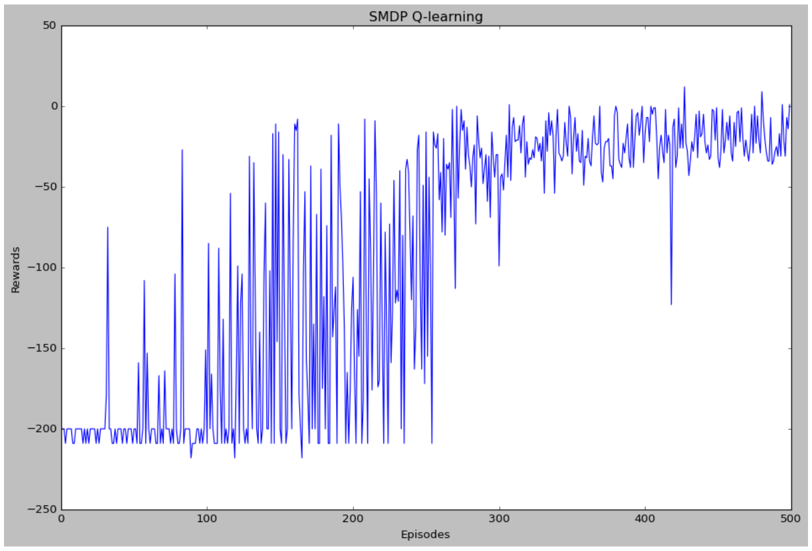

Figure 7: Plot of rewards for SMDP Q-Learning for the given set of options.

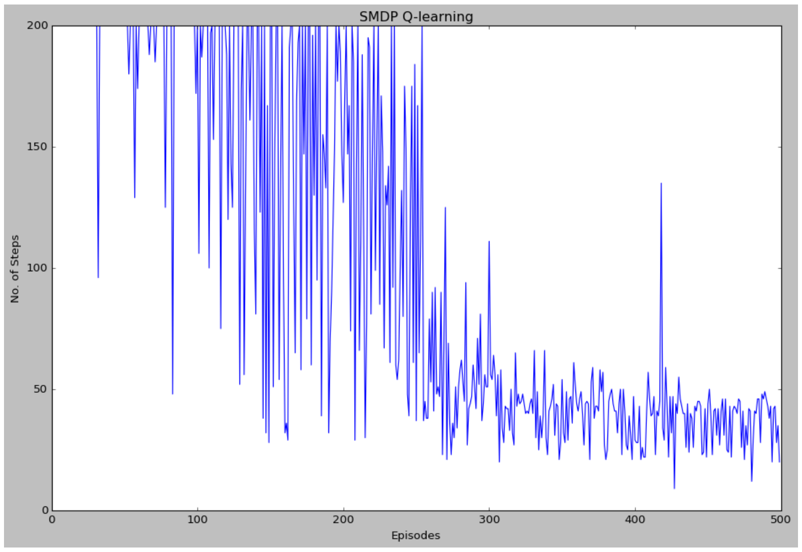

Figure 8: Plot of steps for SMDP Q-Learning for the given set of options.

The policy learnt through SMDP Q-Learning for each of the six options can be visualized in the heatmap shown below:

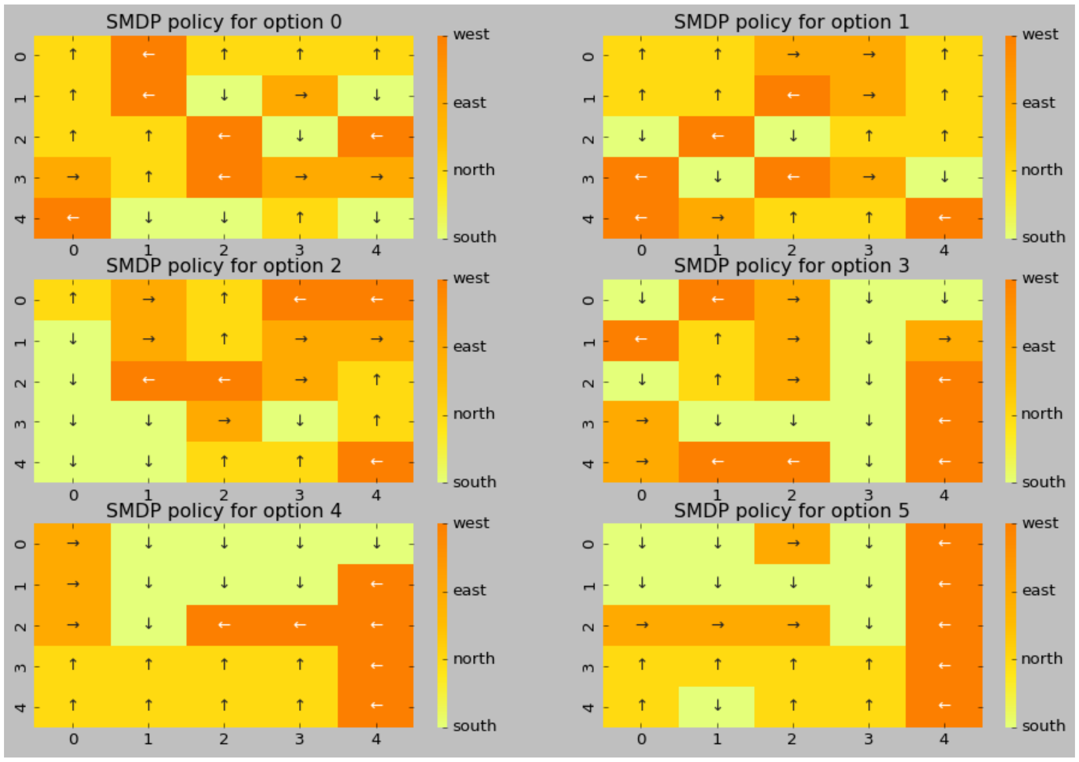

Figure 9: Visualization of the learned policy for each option for SMDP Q-Learning

4.3 Plots for Intra option Q-Learning

The plot of rewards and steps taken as a function of episodes is shown in the graphs below:

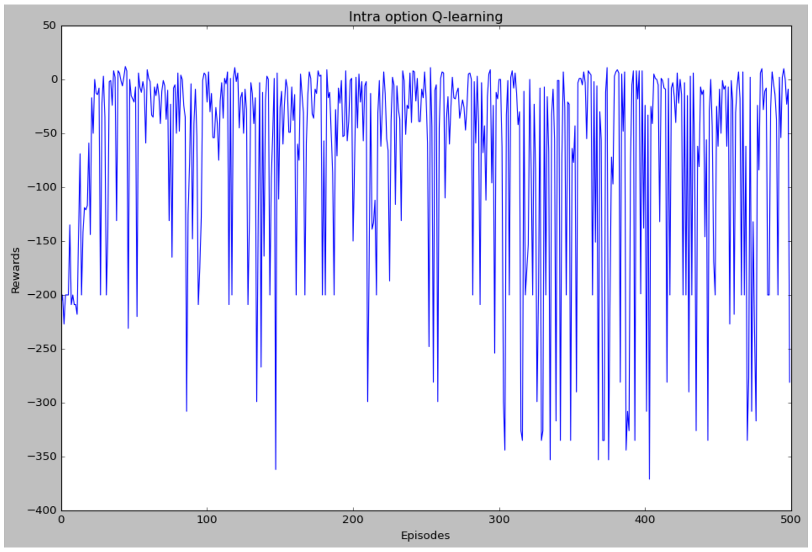

Figure 10: Plot of rewards for Intra option Q-Learning for the given set of options.

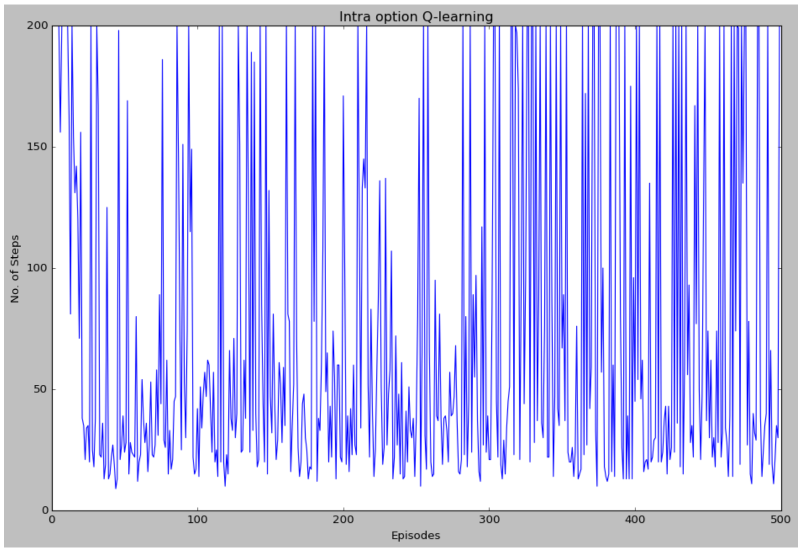

Figure 11: Plot of steps for Intra option Q-Learning for the given set of options.

The policy learnt through Intra option Q-Learning for each of the six options can be visualized in the heatmap shown below:

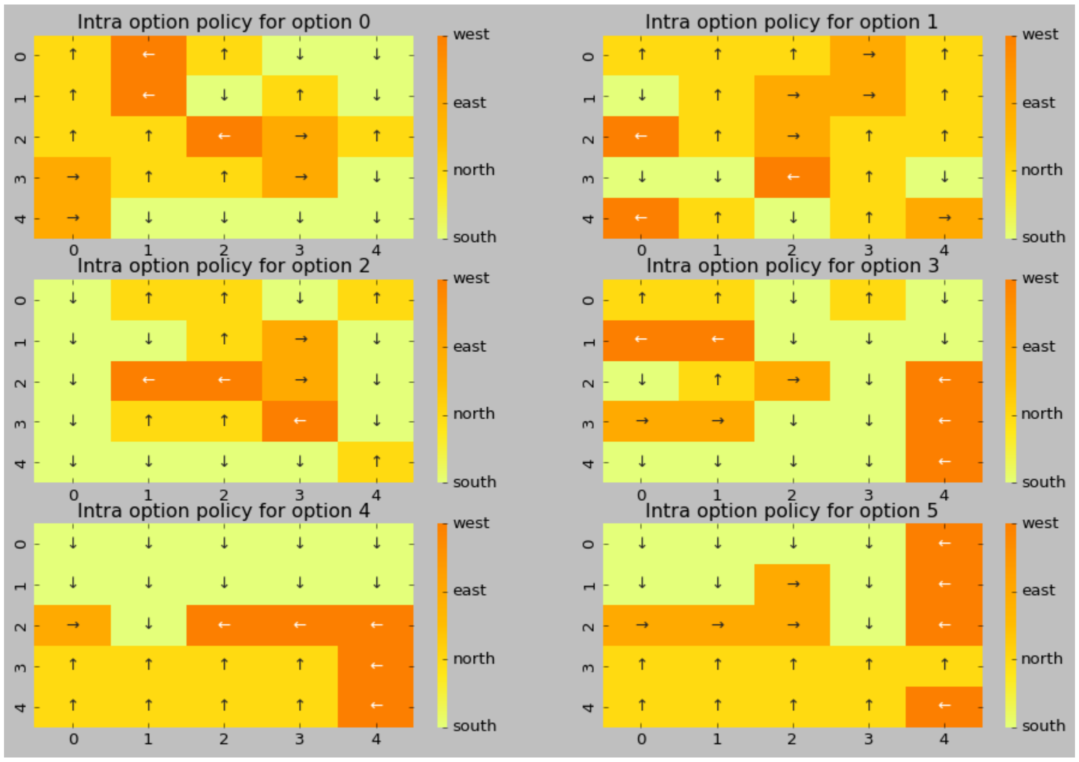

Figure 12: Visualization of the learned policy for each option for Intra option Q-Learning

5 Conclusion

The average rewards obtained by Intra options Q-Learning after solving the environment (which is about 15) is relatively higher than SMDP Q-Learning (about 7.5). This is because in Intra option the frequency of the option value updates is more than in SMDP Q-Learning. And also while the updating Q-value of the selected option, other options which coincides with the current option's primitive actions are also updated simultaneously. This shows that the Intra option method is represented much better at the lower state level which is evident form the fact the rewards obtained by SMDP Q-learning didn't reach the rewards obtained by Intra options even after sufficient number of runs.