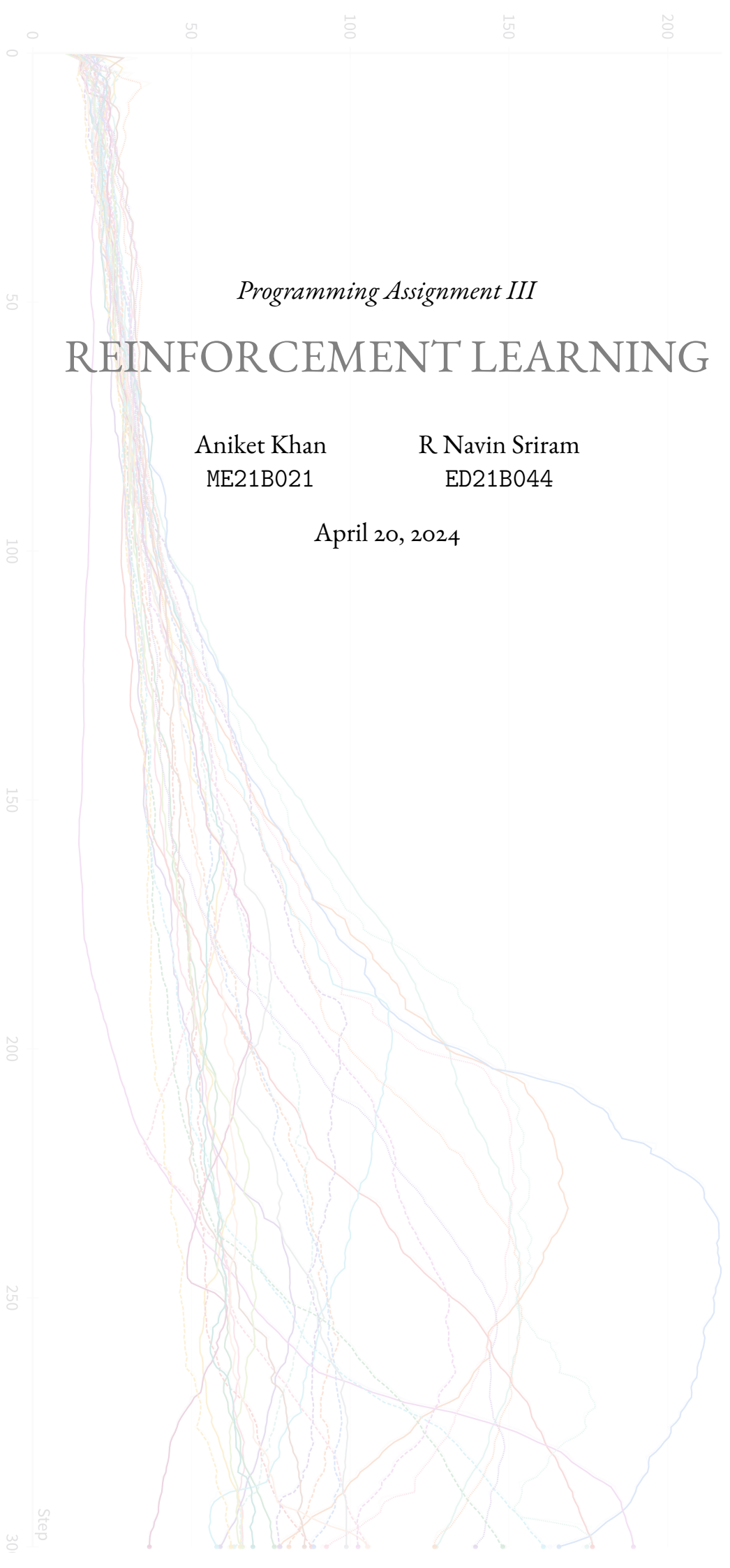
Programming Assignment III

REINFORCEMENT LEARNING

Aniket Khan
ME21B021

R Navin Sriram
ED21B044

April 20, 2024



Contents

1

Structure of the Project

1.1 Introduction

In our project we have explored smdp qlearning and intra option qlearning with various options. We explored fixed policy options, options learned during qlearning and options learned during qlearning with pseudo rewards. We have created a class called SMDP with various functions like plotting Q table and reward curves. The class allows us to change parameters on the fly. We have combined everything into a single project file called SMDP.ipynb.

1.1.1 Tunable Parameters:

- **Learning Rate(α):** Setting an appropriate learning rate ensures that the Q-values converge to their optimal values efficiently without overshooting or oscillating around the true values. If the learning rate is too high, it may lead to unstable and divergent updates, while if it is too low, convergence may be slow or the algorithm may get stuck in suboptimal solutions.
- **Exploit vs Explore(ϵ):** Tuning epsilon in epsilon-greedy policy is crucial as it balances the exploration-exploitation tradeoff, ensuring efficient learning, avoiding local optima, and enabling adaptive behavior in dynamic environments.

Selected Hyper Parameters for both algorithms:

- $\alpha 1$ (for SMDP): 0.1
- $\alpha 2$ (for options): 0.05
- ϵ : 0.15

1.2 Cases:

We made a couple variations of the options and ran the SMDP QLearning and SMDP Intra option Qlearning. All the test cases are as follows:

- SMDP Qlearning with **Learnable options** and **no pseudo reward**
- SMDP Qlearning with **Learnable options** and pseudo reward:
- SMDP Qlearning with **fixed options**
- SMDP Qlearning with **only options** and no primitives
- SMDP Intra Option Qlearning with **Learnable options** with pseudo reward

1.3 Plots

- **Average Episodic Reward vs Episodes** for each algorithms
- **Q tables** for each of the options and the entire problem with frequency of visits.(Note: OY is mistakenly written as oOy in some places)

1.4 Code

- Here is the github repository link for the assignment.

2

Code-Snippets

2.1 SMDP Qlearning:

Action Selection for the Policies

```
def choose_action(state,qs,eps):
    length = np.shape(qs)[1]
    if not qs[state].any():
        return random.randint(0,length-1)
    action = np.argmax(qs[state])

    if np.random.rand() < eps:
        action = np.random.randint(0,length-1)
        return action

    return action

def Red(q_r,state,eps):
    optdone = False
    optact = choose_action(state,q_r,eps)
    state = decode(state)
    if state[0] == R[0] and state[1] == R[1]:
        optdone = True
    return optact,optdone
# similarly for Green, Blue and Yellow(R is a 1x2 array storing
    the goal coordinates)

def Custom(q_val,state,eps):
    optdone = False
    optact = choose_action(state,q_val,eps)
    state = decode(state)
    if state[0] == R[0] and state[1] == R[1]:
        optdone = True
    elif state[0] == G[0] and state[1] == G[1]:
        optdone = True
    elif state[0] == B[0] and state[1] == B[1]:
        optdone = True
    elif state[0] == Y[0] and state[1] == Y[1]:
        optdone = True
    return optact,optdone
```

Options Update

```
reward_bar = 0
if action > 5 :
    count = 0
    optdone = False
    current_state = state
    state = encode(get_state(state)[0:2])
    epsilon = 0.2
    while (optdone == False) :

        if action == 6:
            optact, optdone = Red(self.Q_opt[0],state,epsilon)
            next_state, reward, done, _, _ = env.step(optact)
            next = encode(get_state(next_state)[0:2])

            if optdone and psuedo:
                self.Q_opt[0][state,optact] =
                self.Q_opt[0][state, optact] + ALPHA2* (reward
                + 20 + GAMMA * np.max(self.Q_opt[0][next])
                - self.Q_opt[0][state, optact])

            else:
                self.Q_opt[0][state,optact] =
                self.Q_opt[0][state, optact] + ALPHA2 * (reward
                + GAMMA * np.max(self.Q_opt[0][next, :])-
                self.Q_opt[0][state, optact])

            self.qof[0][state,optact] += 1

    # similarly three other updates for Action = 7,8,9

    reward_bar =  reward_bar + (GAMMA**count)*reward
    ep_reward += reward
    count += 1
    state = next

    if optdone == True:
        self.q[current_state, action] +=
        ALPHA1 * (reward_bar - self.q[current_state, action]
        + (GAMMA**count) * np.max(self.q[next_state, :]))
        state = env.s

    running = 0.05 * ep_reward + (1 - 0.05) * running
    self.avg_reward.append(running)
    self.total_reward.append(ep_reward)
```

Primitive Update

```
steps += 1
epsilon = 0.15
action = choose_action(state,self.q,epsilon)

if action < 6:
    next_state, reward, done, _, _ = env.step(action)
    ep_reward += reward
    self.q[state,action] =
    self.q[state, action] + ALPHA1 * (reward + GAMMA *
    np.max(self.q[next_state, :])- self.q[state, action])
    self.qf[state,action] += 1
    state = next_state
```

2.2 INTRA-OPTION QLEARNING:

Primitive update

```
running = 0
stateg, _ = env.reset()
done = False
ep_reward = 0
steps = 0

while not done :
      steps += 1
      epsilon = 0.15
      action = choose_action(stateg,self.q,epsilon)

      if action < 6:
         next_state, reward, done, _, _ = env.step(action)
         ep_reward += reward
         self.q[stateg,action] =
         self.q[stateg, action] + ALPHA1 * (reward + GAMMA *
         np.max(self.q[next_state, :])- self.q[stateg, action])
         self.qf[stateg,action] += 1
         grid_state = encode(get_state(stateg)[0:2])
         next = encode(get_state(next_state)[0:2])

         for j in range(len(self.Q_opt)):
             if np.argmax(self.Q_opt[j][grid_state]) == action:
                 self.Q_opt[j][grid_state][action] +=
                 ALPHA1 * (reward + GAMMA *
                 np.max(self.Q_opt[j][next]) -
                 self.Q_opt[j][grid_state][action])

         stateg = next_state
```

Options update

```
reward_bar = 0
if action > 5 :

    count = 0
    optdone = False
    current_state = stateg
    state = encode(get_state(stateg)[0:2])
    epsilon = 0.1
    while (optdone == False) :

     if action == 6:
        optact, optdone = Red(self.Q_opt[0],state,epsilon)
        next_state, reward, done, _, _ = env.step(optact)
        next = encode(get_state(next_state)[0:2])

        if optdone and psuedo:
            self.Q_opt[0][state,optact] =
            self.Q_opt[0][state, optact]
            + ALPHA2* (reward + 20
            + GAMMA * np.max(self.Q_opt[0][next])
            - self.Q_opt[0][state, optact])

        else:
            self.Q_opt[0][state,optact] = self.Q_opt[0][state,
               optact] + ALPHA2 * (reward + GAMMA *
            np.max(self.Q_opt[0][next, :]) -
            self.Q_opt[0][state, optact])

        self.qof[0][state,optact] += 1

     reward_bar =  reward_bar + (GAMMA**count)*reward
     ep_reward += reward
     count += 1
     state = next

     U_intra = (1 - optdone_to_beta(optdone)) *
     self.q[stateg, optact] + optdone_to_beta(optdone) *
     np.max(self.q[next_state])
     self.q[stateg, action] += ALPHA1 * (reward -
     self.q[stateg, action] + GAMMA* U_intra)
```

2.3 Fixed Deterministic Policy

Action Maps and a Sample code to run the Experiments

```
action_map1 = np.array([[1,3,3,2,0],
                        [1,3,3,2,0],
                        [1,3,3,2,0],
                        [1,3,2,2,0],
                        [1,3,2,2,2]
                        ])

action_map2 = np.array([[0,3,3,2,1],
                        [0,3,3,2,1],
                        [0,3,3,2,1],
                        [0,3,2,2,1],
                        [0,3,2,2,2]
                        ])

action_map3 = np.array([[2,2,2,2,0],
                        [1,3,1,2,0],
                        [1,3,1,2,0],
                        [1,3,0,2,0],
                        [1,3,3,3,3]
                        ])

action_map4 = np.array([[0,3,2,2,0],
                        [2,1,1,1,3],
                        [1,0,3,1,0],
                        [0,2,2,0,3],
                        [3,1,3,2,1]
                        ])

Q1 = getOptfromMap(action_map1)
Q2 = getOptfromMap(action_map2)
Q3 = getOptfromMap(action_map3)
Q4 = getOptfromMap(action_map4)
Q_opt = np.asarray([Q1, Q2, Q3, Q4])
q_custom = np.zeros((500,10))
qf_custom = np.zeros((500,10))
Qof_custom = np.zeros((4,25,4))

smdp = SMDP(q_custom, qf_custom, Q_opt, Qof_custom)
smdp.reset()
q = smdp.Qlearn_fixed(psuedo=True)
smdp.plot()
smdp.plot_Q_options()
smdp.plot_SMDP()
```

Helper Functions

```
def get_state(state):
    row, col, pass_id, dest_id = env.unwrapped.decode(state)
    state = np.asarray([row,col,pass_id,dest_id])
    return state

def encode(pose):
    [row,col] = pose
    state_num = row * 5 + col
    return state_num

def decode(state_num):
    row = state_num // 5
    col = state_num % 5
    return row, col

def getOptfromMap(action_map):
    Q_opt_custom = np.zeros((25, 4))
    for i in range(5):
        for j in range(5):
            # Get the state number
            state_num = encode([i, j])
            # Get the value at the current grid cell
            value = action_map[i, j]
            # Update the corresponding subarray in the 25x4
              array
            Q_opt_custom[state_num] = np.zeros(4)
            Q_opt_custom[state_num][value] = 1
    return Q_opt_custom
```

Plotting the main Q table

```
def plot_SMDP(self,mode = "learnable"):
    destination = ["R","G","Y","B"]
    passenger = ["R","G","Y","B","Taxi"]
    if mode == "learnable":
        actions =
        ["S", "N", "E", "W","P","D","OR","OG","OB","oOY"]
    elif mode == "just_options":
        actions = ["P", "D","OR","OG","OB","OY"]
    else:
         actions = ["S", "N", "E", "W","P","D","O"]
    fig, axs = plt.subplots(5, 4, figsize=(16, 20))
    fig.suptitle('Q values for all cases')
    for k in range(20):
        if mode == "learnable":
            q_plot = np.zeros((5, 5, 10))
        elif mode == "just_options":
            q_plot = np.zeros((5, 5, 6))
        else:
            q_plot = np.zeros((5, 5, 7))
        q_value = self.q[k::20]
        row, col, pass_id, dest_id = env.unwrapped.decode(k)
        row = k // 4
        col = k % 4
        for i in range(5):
            for j in range(5):
                state_num = encode([i,j])
                q_plot[i, j] = q_value[state_num]

        axs[row][col].set_title(f"Dest {destination[dest_id]},
            Pass {passenger[pass_id]}")
        axs[row][col].pcolor(q_plot.max(-1), edgecolors='k',
            linewidths=2)
        plt.colorbar(axs[row][col].pcolor(q_plot.max(-1)), ax=
            axs[row][col])

        for i in range(5):
            for j in range(5):
                state_num = i * 5 + j
                anno = actions[np.argmax(q_plot[i, j])]
                size = 'large'
                axs[row,col].text(j + 0.5, i + 0.5, anno, ha='
                    center', va='center', color='red', size=size)

        axs[row,col].invert_yaxis()
        plt.title(mode)

    plt.tight_layout()
    plt.show()
```

3

SMDP Qlearning

3.1 SMDP Q learning with learnable Options and no Psuedo Reward

- ϵ-greedy action without decay for Action selection
- α_1 (smdp QL), α_2 (options QL) = 0.1, 0.05
- ϵ_1 (smdp), ϵ_2 (options) = 0.15, 0.1
- No Psuedo Rewards : No incentive given to the agent on completion of an option

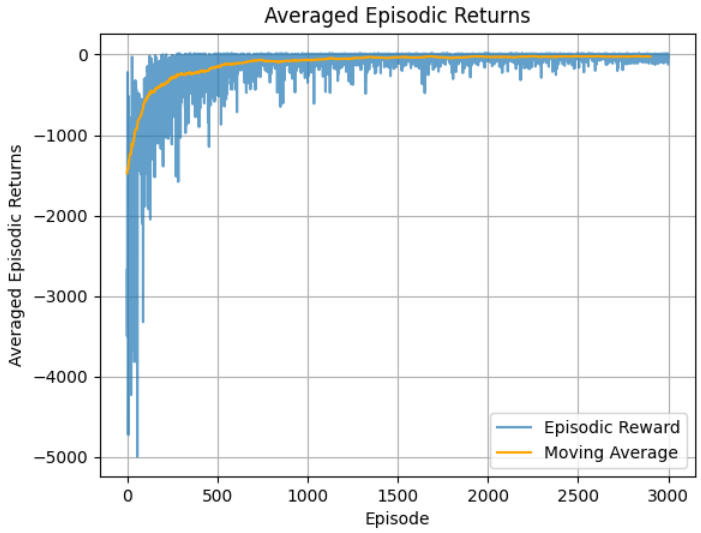


Figure 1: Average Reward vs Episodes (no pseudo reward)

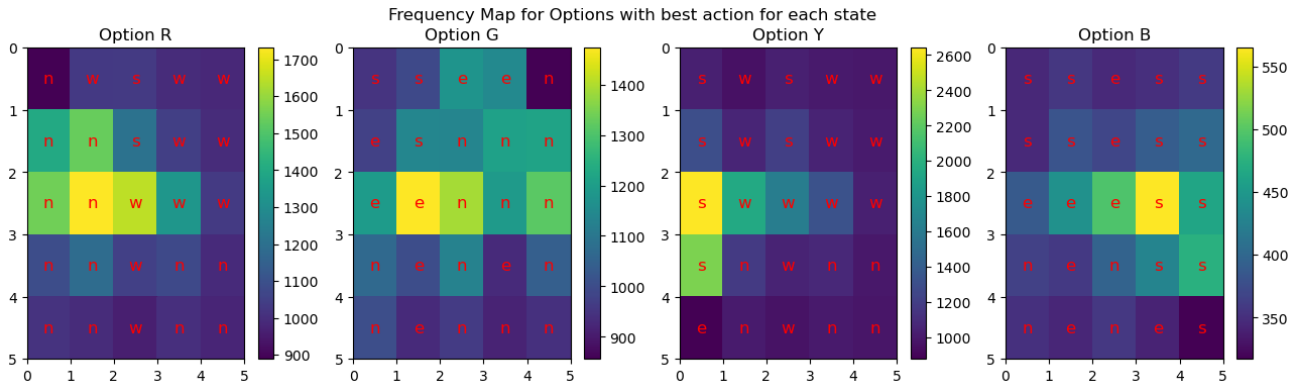


Figure 2: Q table for learnt options

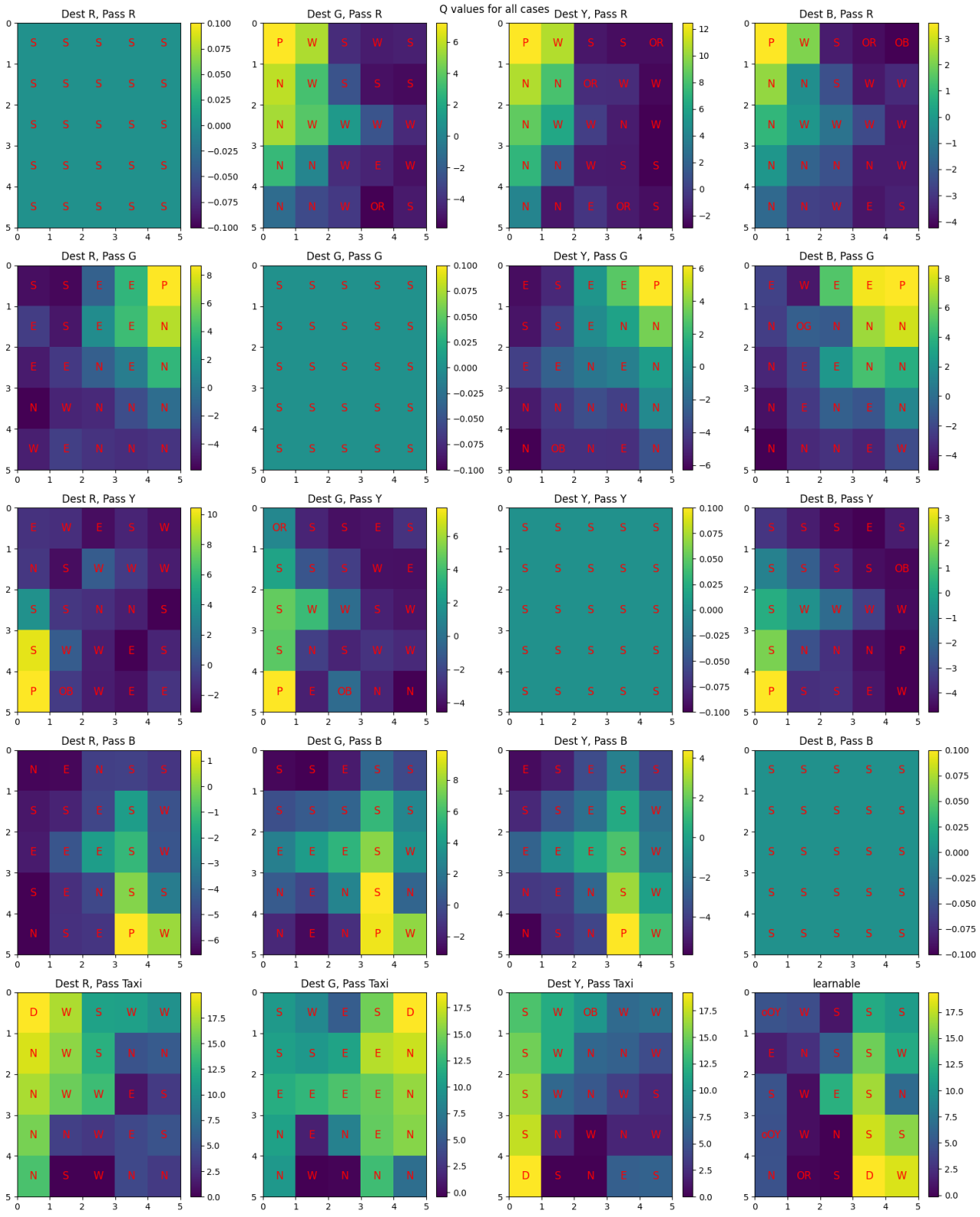


Figure 3: Q table with learnable options (no pseudo reward)

3.2 SMDP Q learning with learnable Options and Psuedo Reward

- ϵ-greedy action without decay for Action selection
- α_1 (smdp QL), α_2 (options QL) = 0.1, 0.05
- ϵ_1 (smdp), ϵ_2 (options) = 0.15, 0.1
- Psuedo Reward : Reward of 20 given on termination of an option

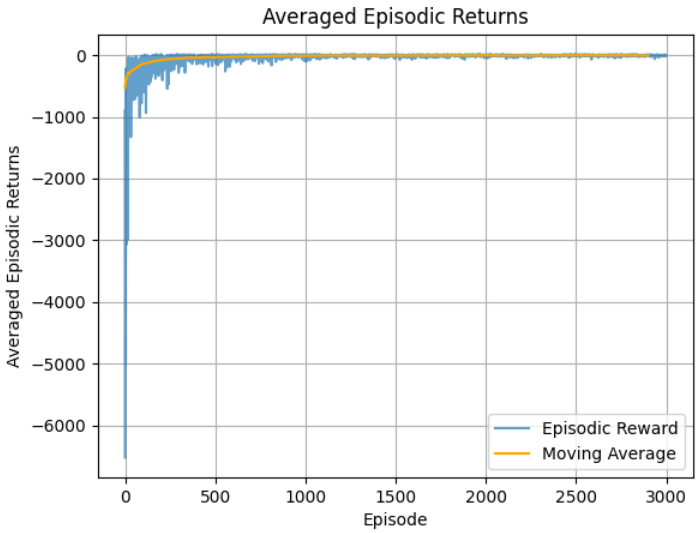


Figure 4: Average Reward vs Episodes (with pseudo reward)

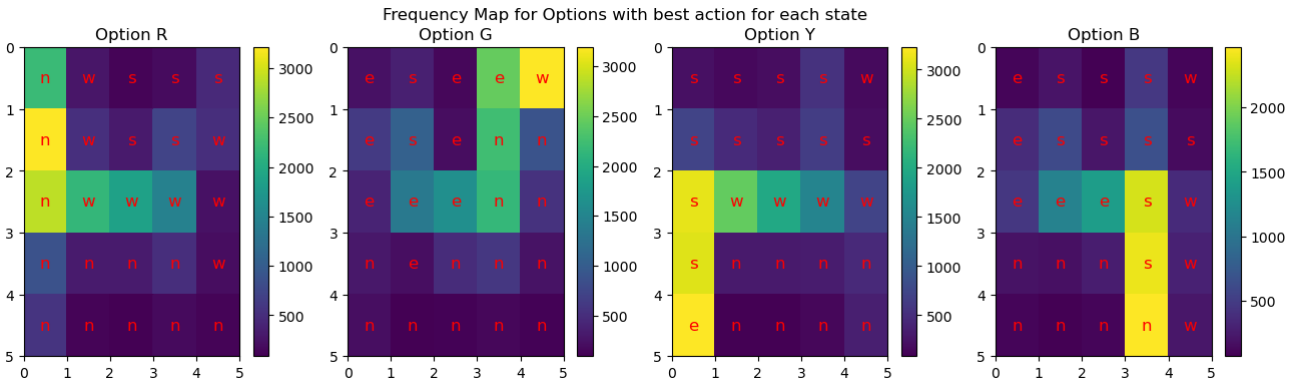


Figure 5: Q table for learnt options

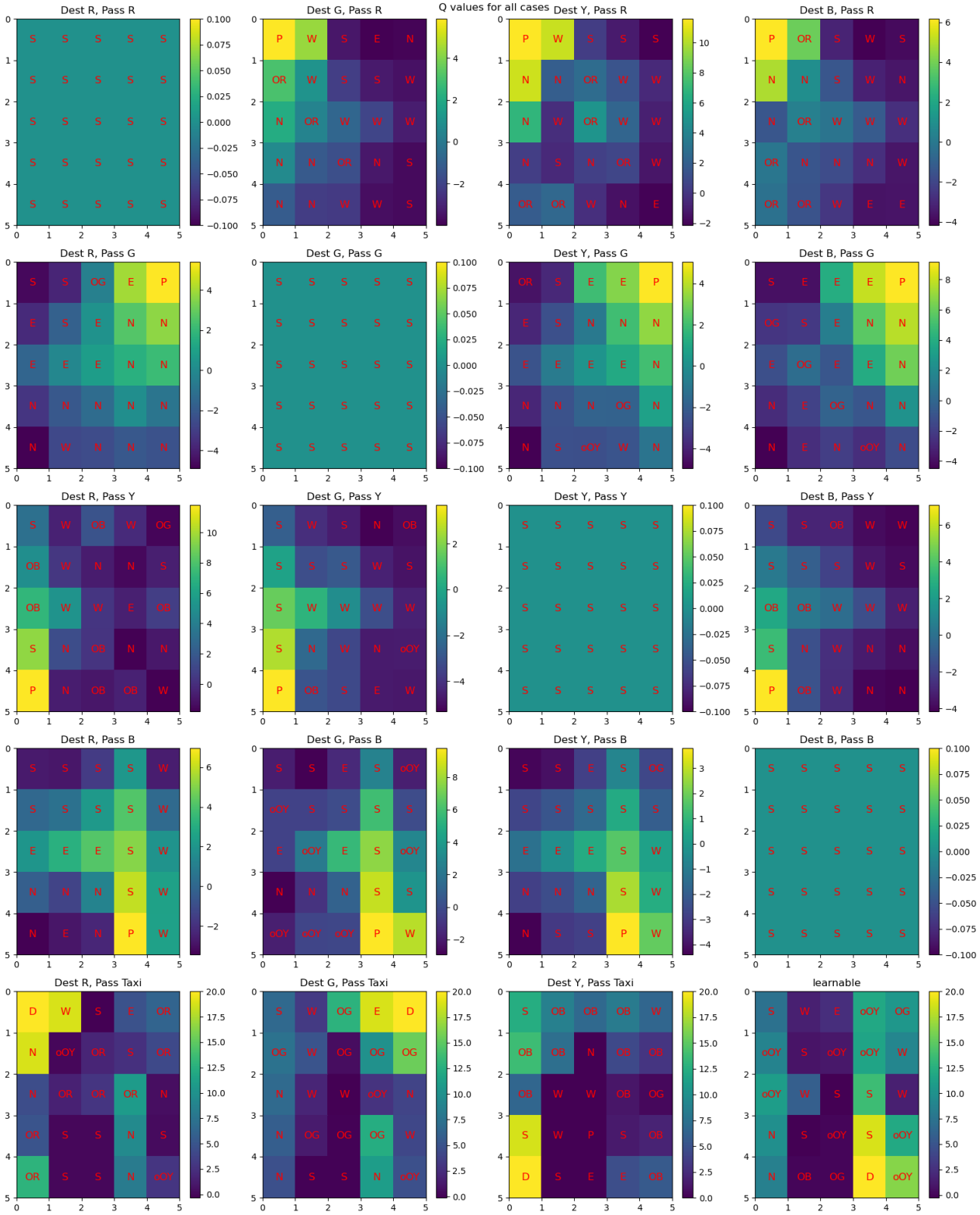


Figure 6: Q Table - Learnable options

3.3 SMDP Q learning with Alternate Fixed Options

- ϵ-greedy action without decay for Action selection
- $\alpha_1 = 0.1$
- $\epsilon_1 = 0.15$
- Four common behvaiours used to create action maps and Four different fixed options

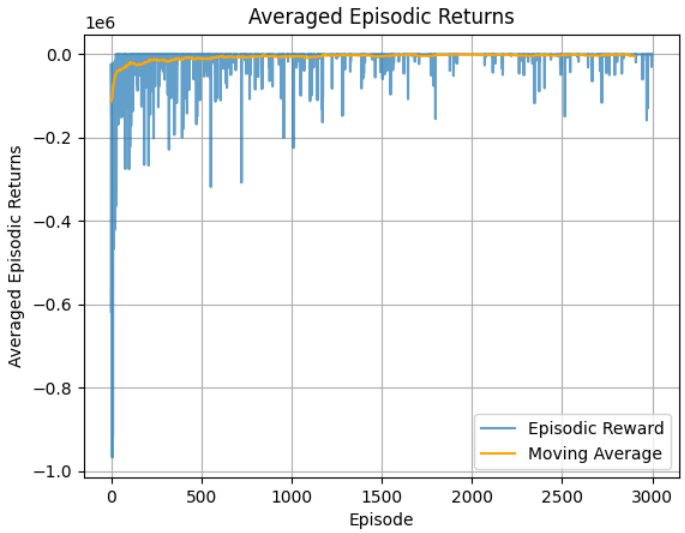


Figure 7: Average Reward vs Episodes with fixed option

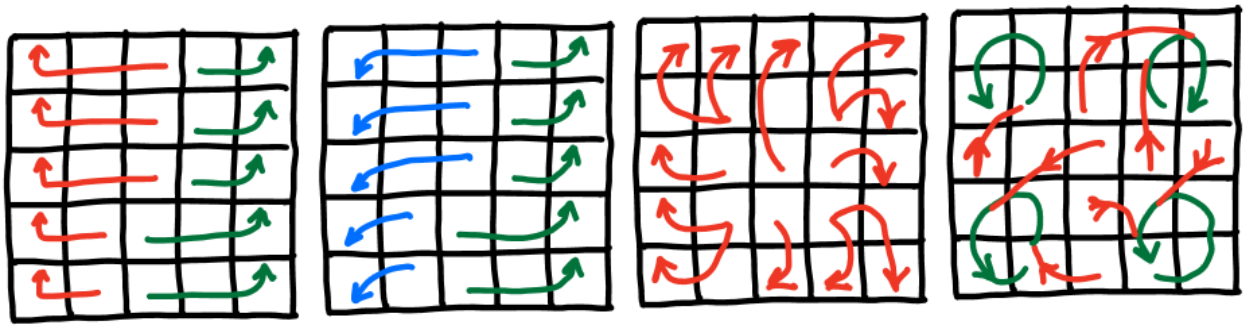


Figure 8: Behaviour of the Four policies

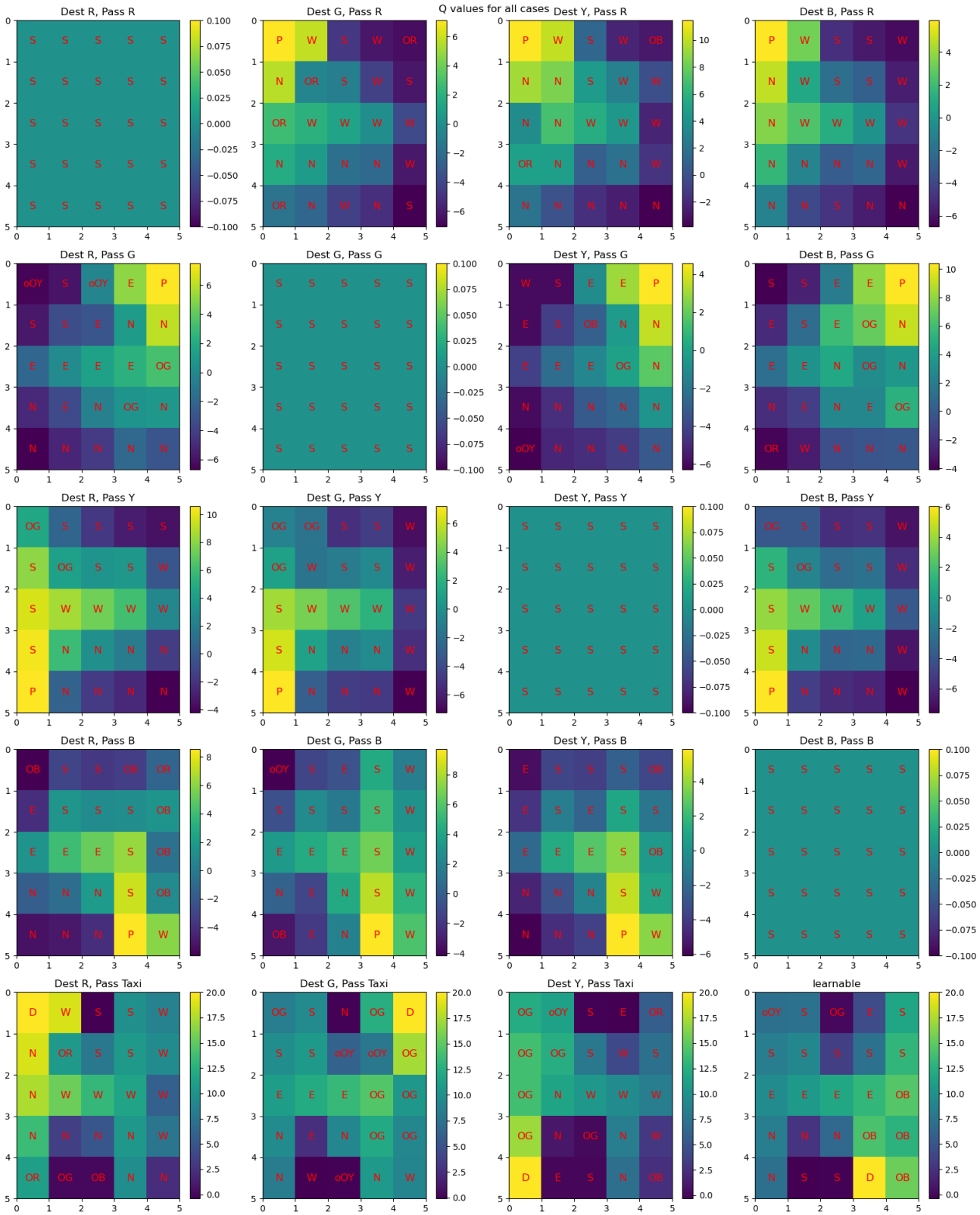


Figure 9: Q table for all primive actions and Fixed options

3.4 SMDP Q learning with only learnable options and no primitive actions

- ϵ-greedy action without decay for Option action selection
- $\alpha_2 = 0.1$
- $\epsilon_2 = 0.15$
- Four learnable options to go to R, G, B, Y

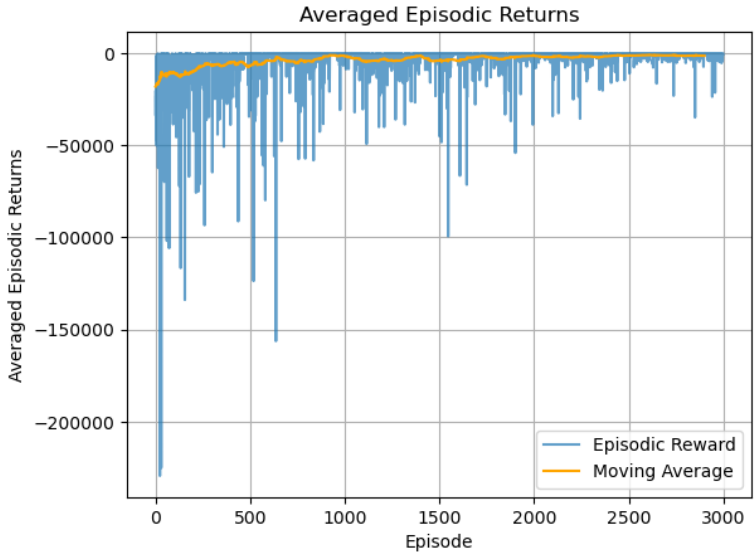


Figure 10: Average Reward vs Episodes with only learnable options

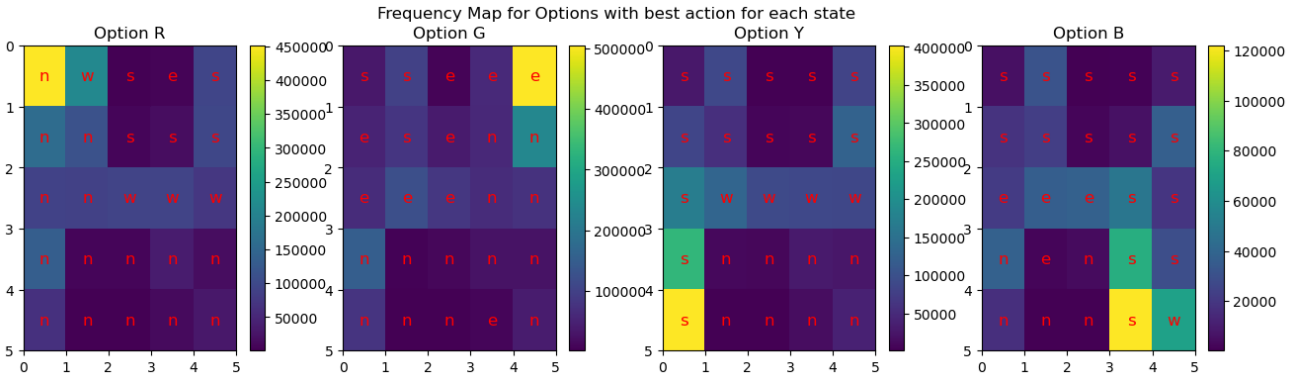


Figure 11: Q tables for learnt options

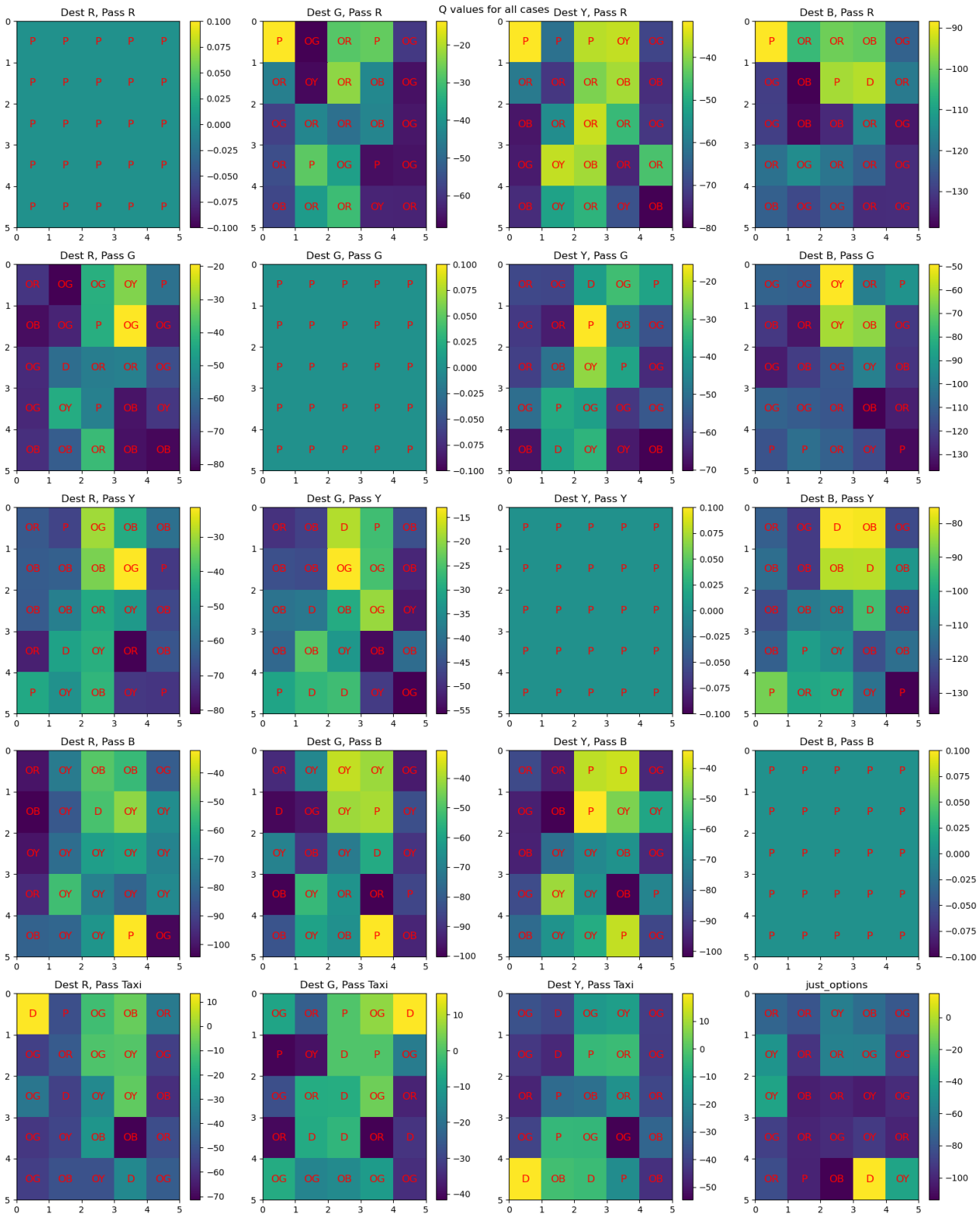


Figure 12: Q table while using only 4 learnable options

3.5 Intra-Option Qlearning

- ϵ-greedy action without decay for Action selection
- α_1 (Intra QL), α_2 (options QL) = 0.1, 0.05
- ϵ_1 (Intra), ϵ_2 (options) = 0.15, 0.1
- No Psuedo Rewards : No incentive given to the agent on completion of an option

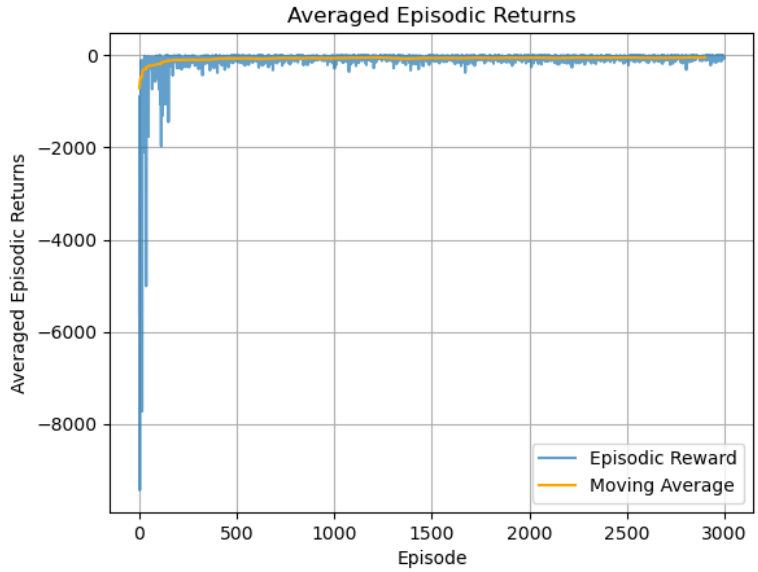


Figure 13: Average Reward vs Episodes with learnable options

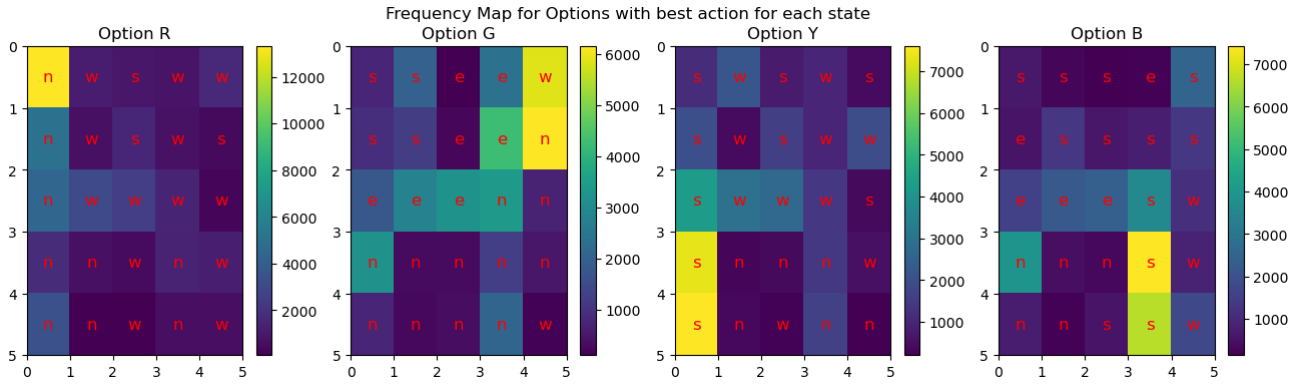


Figure 14: Learnt Options' Q tables

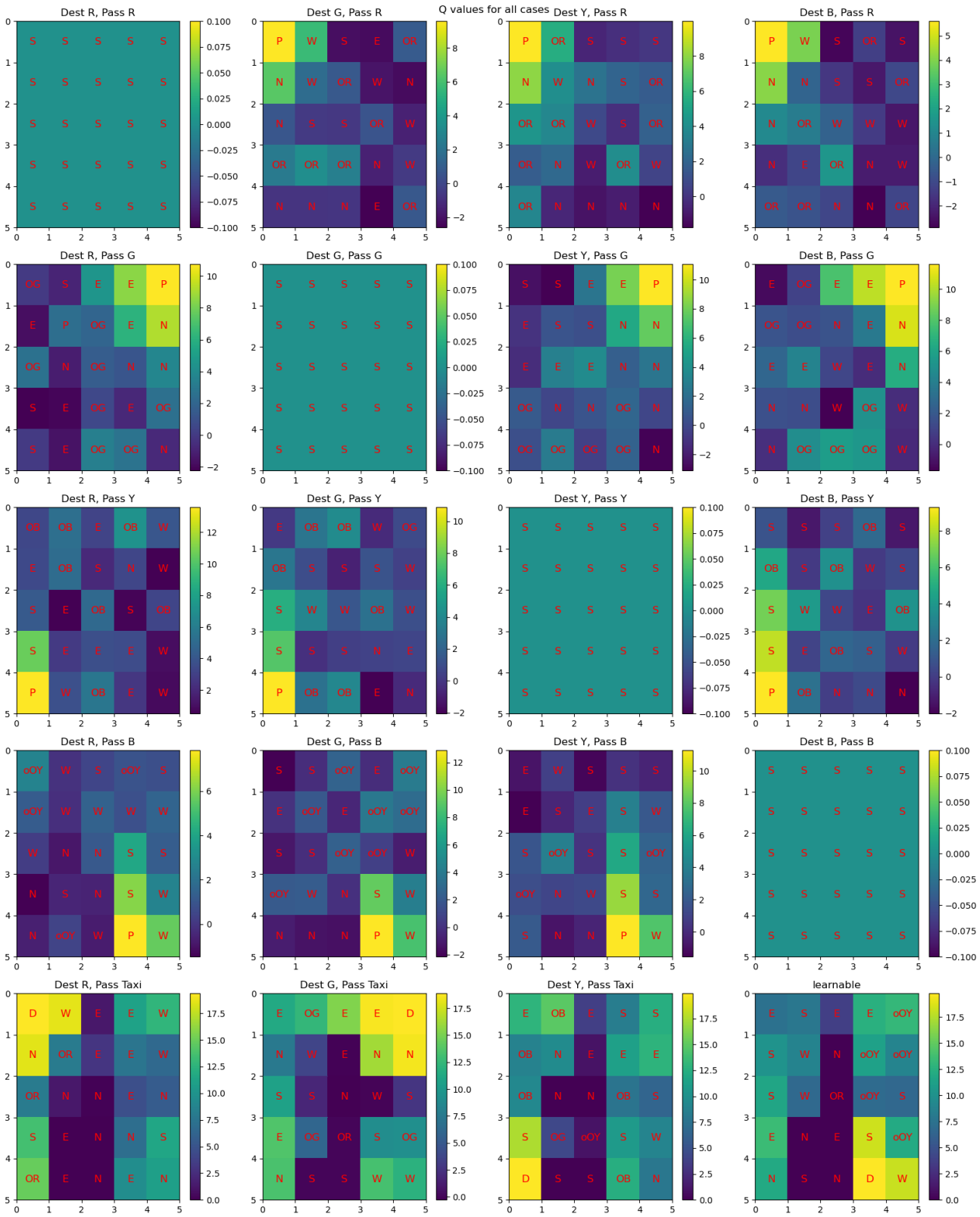


Figure 15: Final Q Table

3.6 Intra-Option Qlearning with Alternate Fixed Options

- ϵ-greedy action without decay for Action selection
- α_1 (Intra QL), α_2 (options QL) = 0.1, 0.05
- ϵ_1 (Intra), ϵ_2 (options) = 0.15, 0.1
- No Psuedo Rewards : No incentive given to the agent on completion of an option

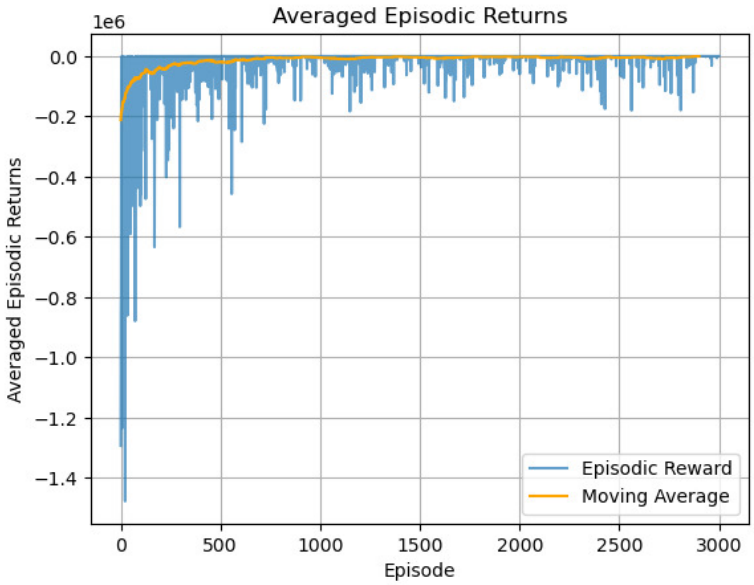


Figure 16: Average Reward vs Episodes with fixed options

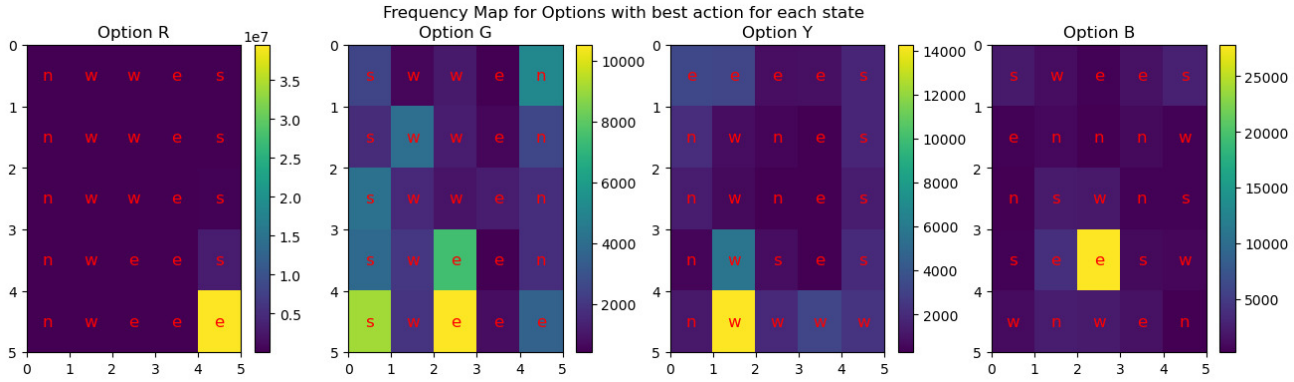


Figure 17: Fixed Options' Q tables

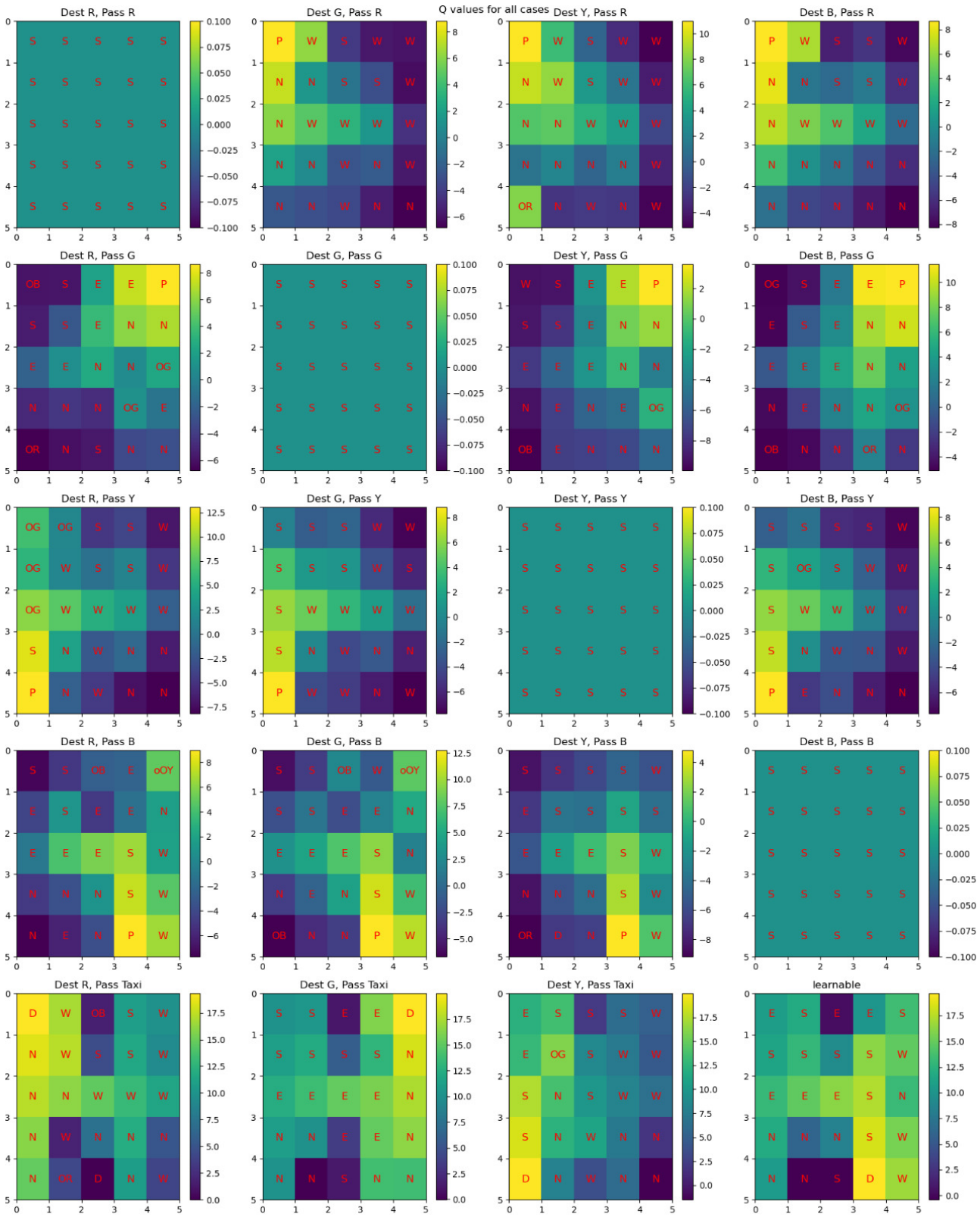


Figure 18: Final Q Table

4

Inference and Conclusion

In our assignment, we went through 2 algorithms, SMDP Qlearning and SMDP intra option Qlearning. We further did some variations of the algorithm by using options with no pseudo rewards, options with pseudo rewards, some mutually exclusive option definitions and using only options to solve the SMDP(learning it on the go). We made some inferences from the reward plots and the q table heatmap as follows:

- The trade off between choosing an option and a primitive action is determined by the pseudo reward (20 in our case), if it is higher it would tend to choose more options.
- **SMDP, No psuedo rewards** : In the case with no pseudo rewards, options were not learnt properly, hence the agent was not able to navigate throught the option properly, which ultimately resulted in the options action being less used. This emphasises **incentivizing the termination** of an option while learning that option.
- Without this, only primitive actions are used and this causes a **lot of oscillations** in steady state due to the small explorative behaviour while selecting each and every action.
- As seen from the heatmap, for no pseudo reward, the frequency of update is scattered all around the table this is due to the fact proper q table was not learnt. This issue is solved by giving a pseudo reward for completing an option.
- **SMDP, with psuedo rewards :** In the case with pseudo rewards, the agent was rewarded for successful termination of the option and hence the agent explored options more and resulted better and more stable curve with rewards almost reaching 9 after being run for 3000 episodes.
- Convergence and Std deviation, both are better when compared to the first case.
- **Fixed Options :**On observing the Q tables in the previous cases, we formulated **4 options based on common behaviours** observed while going to any destination, keeping it fixed. Firstly, the time to convergence and to run the algorithm is very long.
- The reward curves are very noisy due to the **largely non-perfect** policies being chosen and the agent being stuck in insanely long episodes without terminations due to the **deterministic nature** and **Non-Mutual exclusiveness**. To tackle this, we **added stochasticity** in the form of a small epsilon value while selecting actions from the options leading to some improvement. The rewards remain sub-par.
- **Only options :**For SMDP with just the options, RGBY, and pick/drop, The resulting rewards are very bad. Intra option update was used in this case. From the q table we can see that the action **"pick" and "drop"** being executed recklessly, even though they result in a high negative reward, highlighting the bad Q values associated with each option. The time taken to run is very high. Given a very long training period, owing to the intra-option nature, it should eventually be able to figure out the optimal policy,

given the option policies are mutually exclusive.

- **Intra-Option :**For intra options, both the run-time and time to convergence is very less. The frequency of updates is very high compared to normal SMDP qlearning, this is because in intra options updates are done while the option is still running, This leads to the best converging policy with pretty stable rewards.
- Intra-option updates lead to the options being preffered more often than primitive actions as compared to normal SMDP learning due to better generalization.

4.1 SMDP Qlearning:

4.1.1 Policy Description:

- The learnt policy for most of the different cases prefers the learnable options to go to R,G,B and Y over primitive actions near the corresponding Passenger location (if not in the taxi already). Each Policy has atleast 4-5 taxi positions where options are executed over actions. In some cases where the Passenger is at B, none of the options are preferred.
- For the alternate set, the fixed options appear less frequently as compared to learnable options. The fixed options are not perfect and not optimal whereas, in the first case the learned options reach near optimality.

4.1.2 Reason:

SMDP Qlearning learns the policy at the level of options, it does not update the main Q table while the option is running, so it entirely depends on how good the options final reward is and the effect of the primitive actions that were used to reach that state are neglected.

4.2 Intra option Qlearning:

4.2.1 Policy Description:

- The policy learnt through Intra option has increased usage of Options across all the states, over twice to that of SMDP. In the case of Fixed Options, the options are rarely used, since they are fixed, the update may or may not improve the usage of options over actions since the options are not very perfect.

4.2.2 Reason:

The increased usage with Learnable Options is due to every-step update increasing exploration, leading to better generalization. The each step update lets the main Q table updated better for faster convergence and more options utilisation.

4.2.3 SMDP v/s Intra option

- **Learnable :** For Learnable options, intra-option converges quicker as compared to SMDP due its exploration within the context of each option. This focused exploration allows the agent to explore state-space more efficiently with respect to specific sub-goals. It is noiser than smdp due to the same reason.
- **Fixed :** The Rewards are is almost similar if not slightly better in the case of intra-option due its focus on learning policies within fixed options.