CS6700: Reinforcement Learning

PA3 EE21S048 NS24Z111

April 20, 2024

Assignment 3

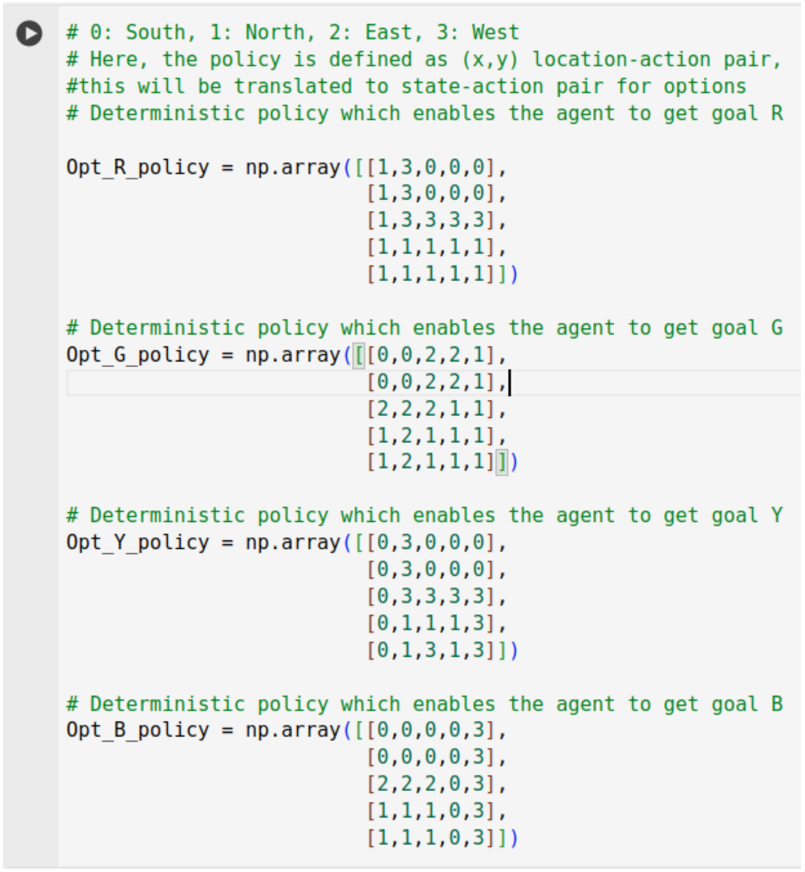
Team Members:

- Matcha Naga Gayathri **(EE21S048)**
- Ragu B **(NS24Z111)**

OPTIONS:

Implementation Details:

- Primitive options are special cases of options that last exactly one step.
- Primitive options are South, North, East, West, Pick, and Drop.
- The rest of the options defined are multi-step, i.e., temporally extended.
- Temporally extended options are Opt_R, Opt_G, Opt_Y, Opt_B.
- These options correspond to [0,1,2,3,4,5,6,7,8,9] respectively.
- Deterministic policies for the set of temporally extended options (`Opt_R`, `Opt_G`, `Opt_Y`, and `Opt_B`) are:



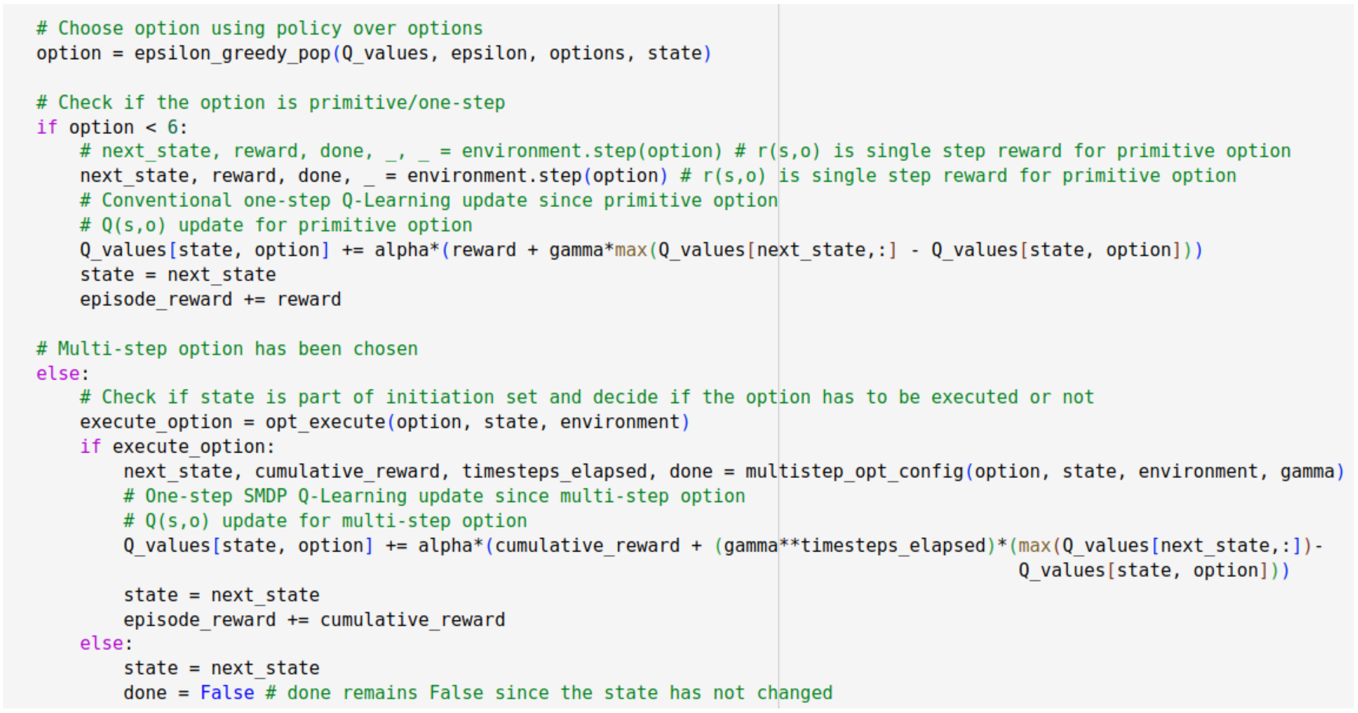
```
# 0: South, 1: North, 2: East, 3: West
# Here, the policy is defined as (x,y) location-action pair,
#this will be translated to state-action pair for options
# Deterministic policy which enables the agent to get goal R

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

# Deterministic policy which enables the agent to get goal G
Opt_G_policy = np.array([[0,0,2,2,1],
                         [0,0,2,2,1],
                         [2,2,2,1,1],
                         [1,2,1,1,1],
                         [1,2,1,1,1]])

# Deterministic policy which enables the agent to get goal Y
Opt_Y_policy = np.array([[0,3,0,0,0],
                         [0,3,0,0,0],
                         [0,3,3,3,3],
                         [0,1,1,1,3],
                         [0,1,3,1,3]])

# Deterministic policy which enables the agent to get goal B
Opt_B_policy = np.array([[0,0,0,0,3],
                         [0,0,0,0,3],
                         [2,2,2,0,3],
                         [1,1,1,0,3],
                         [1,1,1,0,3]])
```

SMDP Q-Learning:

- SMDP (Semi-Markov Decision Process) Q-Learning extends traditional Q-Learning to handle options, which are temporally extended actions. Options allow the agent to select a course of action that may take multiple time steps to complete.
- In SMDP Q-Learning, the Q-values represent the expected return when starting in a particular state and taking a specific action or option. Updates to Q-values occur after the completion of each action or option, considering the cumulative rewards obtained during its execution.

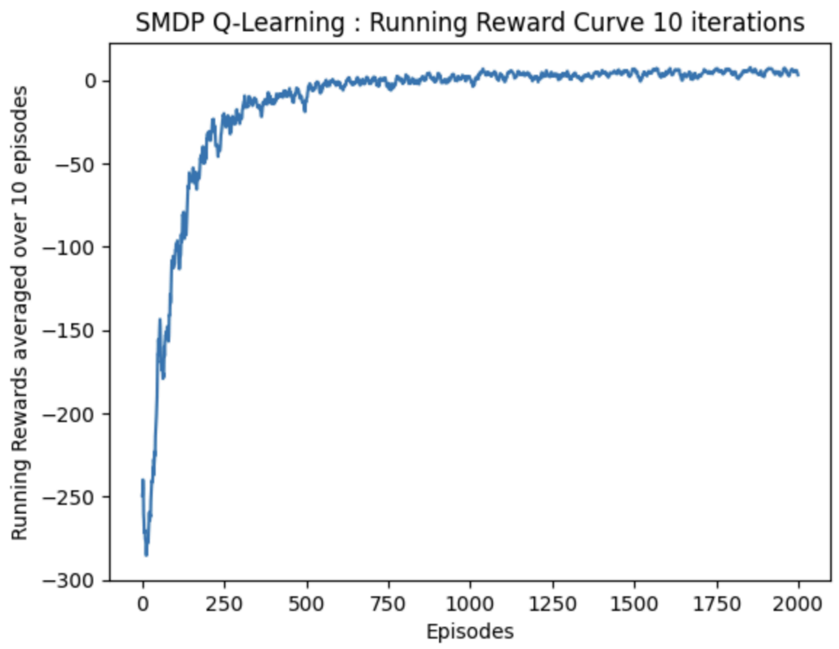
The SMDP version of one-step Q-learning

$$Q(s,o) \leftarrow Q(s,o) + \alpha \left[r + \gamma^k \max_{a \in \mathcal{O}} Q(s',a) - Q(s,o) \right],$$



```
# Choose option using policy over options
option = epsilon_greedy_pop(Q_values, epsilon, options, state)

# Check if the option is primitive/one-step
if option < 6:
    # next_state, reward, done, _, _ = environment.step(option) # r(s,o) is single step reward for primitive option
    next_state, reward, done, _ = environment.step(option) # r(s,o) is single step reward for primitive option
    # Conventional one-step Q-Learning update since primitive option
    # Q(s,o) update for primitive option
    Q_values[state, option] += alpha*(reward + gamma*max(Q_values[next_state,:] - Q_values[state, option]))
    state = next_state
    episode_reward += reward

# Multi-step option has been chosen
else:
    # Check if state is part of initiation set and decide if the option has to be executed or not
    execute_option = opt_execute(option, state, environment)
    if execute_option:
        next_state, cumulative_reward, timesteps_elapsed, done = multistep_opt_config(option, state, environment, gamma)
        # One-step SMDP Q-Learning update since multi-step option
        # Q(s,o) update for multi-step option
        Q_values[state, option] += alpha*(cumulative_reward + (gamma**timesteps_elapsed)*(max(Q_values[next_state,:])-
                                                                                  Q_values[state, option]))

        state = next_state
        episode_reward += cumulative_reward
    else:
        state = next_state
        done = False # done remains False since the state has not changed
```

Reward curve Plot:



Q-table:

	South	North	East	West	Pick	Drop	Opt_R	Opt_G	Opt_Y	Opt_B
0	-0.721105	-0.662189	-0.881495	-0.815337	-6.901653	-8.859009	0.0	-2.088410	-1.916754	-1.844331
1	-0.095171	2.229510	0.242567	1.164757	7.895817	-7.985333	0.0	-10.230138	-4.208514	-8.717181
2	1.896934	3.823296	2.097679	6.640128	15.910837	-3.961125	0.0	-6.957582	-0.353043	-6.024710
3	0.438168	2.901605	0.123421	2.992622	9.679705	-6.364480	0.0	-8.973413	-3.257961	-8.785073
4	-2.102090	-1.111058	-2.182823	-1.110207	-11.108888	-11.104766	0.0	-6.053827	-5.234522	-10.192922

Inference:

- From the Q-table, state 1, the taxi and the passenger are both at location R, while the passenger's destination is at location G. In this case, it's logical for the taxi to pick up the passenger, as confirmed by examining the Q-table for state 1. Notably, the Q-value of the "Pick" option is the highest compared to other options. Additionally, it's observed that the Q-value for the option "Opt_R" is 0, indicating that this option does not execute in this particular state.
- From the reward plot, we can see that, with the increase in the number of episodes, the reward converges, and the Q-values are close to the optimal values.
- It is to be noted that the Q-values are not zero in state 0; instead, if the agent has to pick the maximum value, it would be Option R. This action will not be taken since it is already in R.

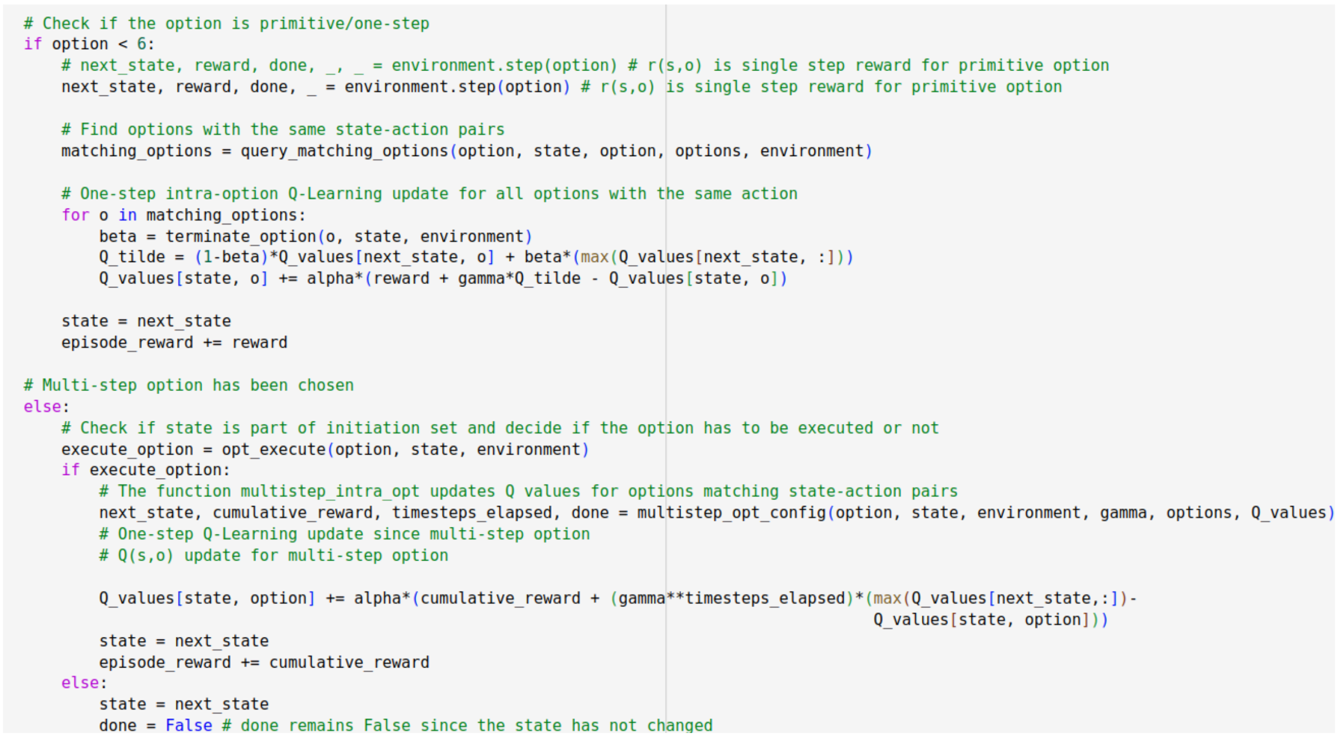
Intra-Option Q-Learning:

- Intra-option Q-Learning focuses on learning within options. Instead of treating options as atomic actions, it decomposes them into primitive actions and learns policies within each option.
- In Intra-Option Q-Learning, the agent maintains separate Q-values for each primitive action within an option. These Q-values are updated during the execution of the option, allowing the agent to learn the value of selecting each primitive action within the context of the option.

Off-policy one-step temporal-difference update is given by:

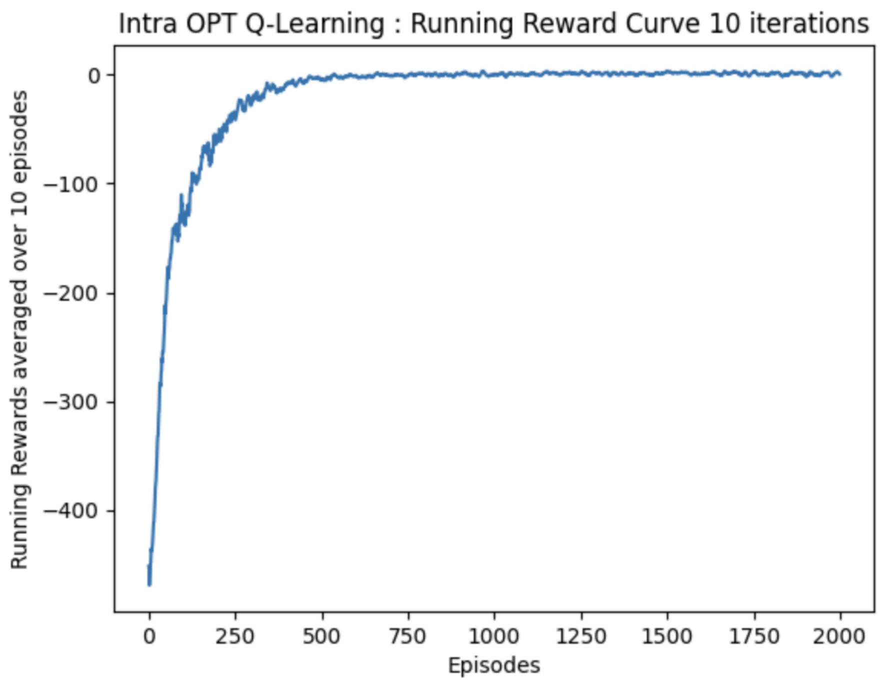
$$Q(s_t, o) \leftarrow Q(s_t, o) + \alpha\Big[\big(r_{t+1} + \gamma \tilde{Q}^*_{\mathcal{O}}(s_{t+1}, o)\big) - Q(s_t, o)\Big],$$

$$\tilde{Q}^*_{\mathcal{O}}(s, o) = (1 - \beta(s))Q^*_{\mathcal{O}}(s, o) + \beta(s) \max_{o' \in \mathcal{O}} Q^*_{\mathcal{O}}(s, o'),$$

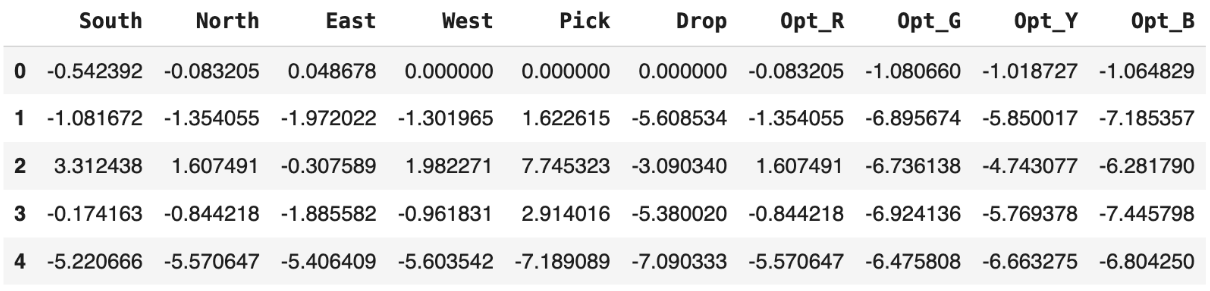


```
# Check if the option is primitive/one-step
if option < 6:
    # next_state, reward, done, _, _ = environment.step(option) # r(s,o) is single step reward for primitive option
    next_state, reward, done, _ = environment.step(option) # r(s,o) is single step reward for primitive option

    # Find options with the same state-action pairs
    matching_options = query_matching_options(option, state, option, options, environment)

    # One-step intra-option Q-Learning update for all options with the same action
    for o in matching_options:
        beta = terminate_option(o, state, environment)
        Q_tilde = (1-beta)*Q_values[next_state, o] + beta*(max(Q_values[next_state, :]))
        Q_values[state, o] += alpha*(reward + gamma*Q_tilde - Q_values[state, o])

    state = next_state
    episode_reward += reward

# Multi-step option has been chosen
else:
    # Check if state is part of initiation set and decide if the option has to be executed or not
    execute_option = opt_execute(option, state, environment)
    if execute_option:
        # The function multistep_intra_opt updates Q values for options matching state-action pairs
        next_state, cumulative_reward, timesteps_elapsed, done = multistep_opt_config(option, state, environment, gamma, options, Q_values)
        # One-step Q-Learning update since multi-step option
        # Q(s,o) update for multi-step option

        Q_values[state, option] += alpha*(cumulative_reward + (gamma**timesteps_elapsed)*(max(Q_values[next_state,:])-
                                                                                      Q_values[state, option]))

        state = next_state
        episode_reward += cumulative_reward
    else:
        state = next_state
        done = False # done remains False since the state has not changed
```

Reward curve Plot:



Q-table:



	South	North	East	West	Pick	Drop	Opt_R	Opt_G	Opt_Y	Opt_B
0	-0.542392	-0.083205	0.048678	0.000000	0.000000	0.000000	-0.083205	-1.080660	-1.018727	-1.064829
1	-1.081672	-1.354055	-1.972022	-1.301965	1.622615	-5.608534	-1.354055	-6.895674	-5.850017	-7.185357
2	3.312438	1.607491	-0.307589	1.982271	7.745323	-3.090340	1.607491	-6.736138	-4.743077	-6.281790
3	-0.174163	-0.844218	-1.885582	-0.961831	2.914016	-5.380020	-0.844218	-6.924136	-5.769378	-7.445798
4	-5.220666	-5.570647	-5.406409	-5.603542	-7.189089	-7.090333	-5.570647	-6.475808	-6.663275	-6.804250

Inference:

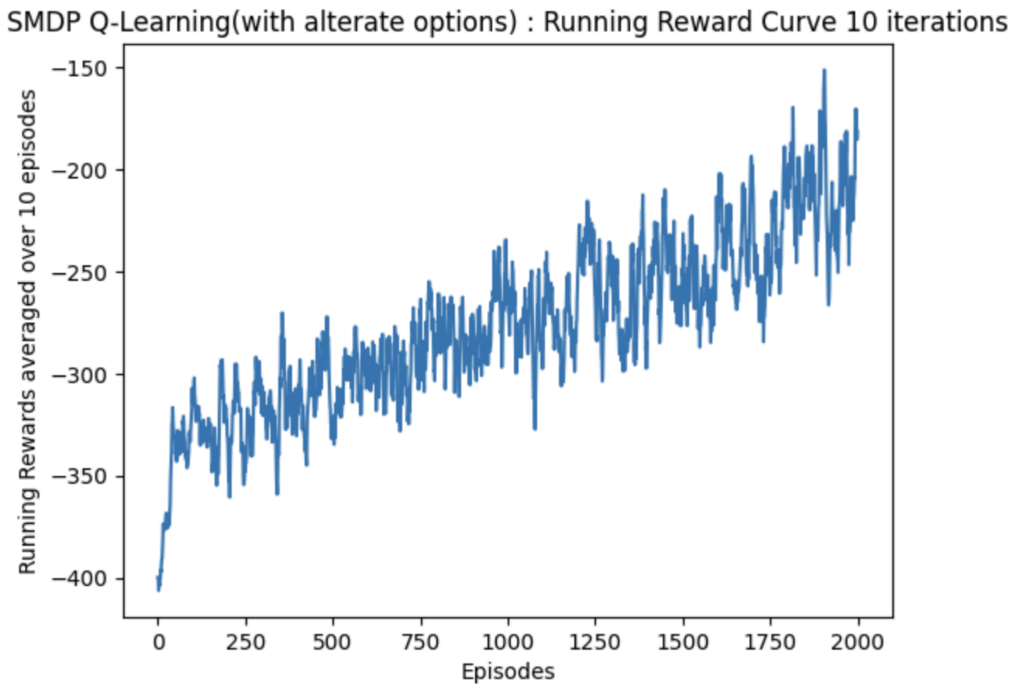
- In state 0, all the values of the Q-table are observed to be close to zero. Over a greater number of episodes, these values converge towards zero, indicating that in this state, the taxi doesn't need to take any action. This suggests that the passenger is already at their destination, making this state invalid for learning or solving.
- In state 1, it can be seen that the option Pick has the highest Q-value, which is to be expected. In State 2, the same trend is noted.

Alternative Set of Options:

The multi-step options have been replaced by four new options, each with a straightforward deterministic policy. These options, namely Opt_S, Opt_N, Opt_E, and Opt_W correspond to movements south, north, east, and west, respectively, for each location in the environment. These definitions can be found in the code cell given below.

```python
# Deterministic policy to always move South
Opt_S_policy = np.array([[0,0,0,0,0],
                         [0,0,0,0,0],
                         [0,0,0,0,0],
                         [0,0,0,0,0],
                         [0,0,0,0,0]])

# Deterministic policy to always move North
Opt_N_policy = np.array([[1,1,1,1,1],
                         [1,1,1,1,1],
                         [1,1,1,1,1],
                         [1,1,1,1,1],
                         [1,1,1,1,1]])

# Deterministic policy to always move East
Opt_E_policy = np.array([[2,2,2,2,2],
                         [2,2,2,2,2],
                         [2,2,2,2,2],
                         [2,2,2,2,2],
                         [2,2,2,2,2]])

# Deterministic policy to always move West
Opt_W_policy = np.array([[3,3,3,3,3],
                         [3,3,3,3,3],
                         [3,3,3,3,3],
                         [3,3,3,3,3],
                         [3,3,3,3,3],])

alternate_options = ["South", "North", "East", "West", "Pick", "Drop",
           "Opt_South", "Opt_North", "Opt_East", "Opt_West"]
```

SMDP Q-Learning:

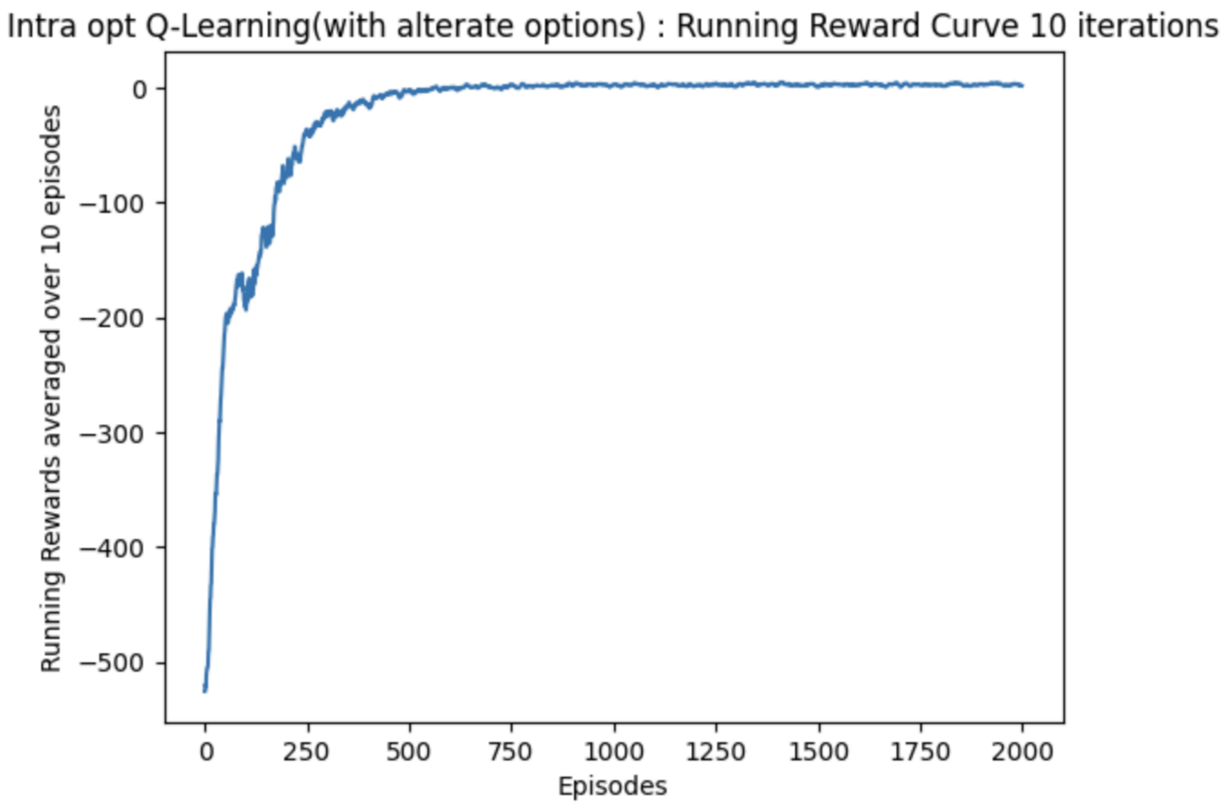


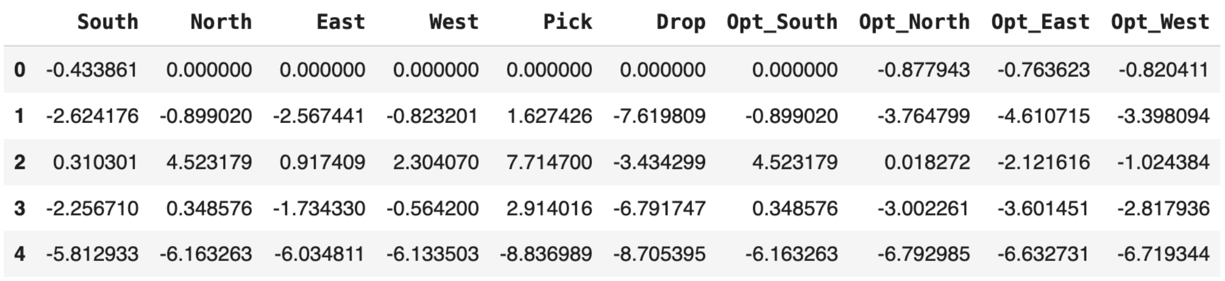
	South	North	East	West	Pick	Drop	Opt_South	Opt_North	Opt_East	Opt_West
0	-0.591898	-0.788870	-0.581400	-0.475677	-3.450540	-4.552740	0.0	-0.439757	-0.430564	-0.504111
1	-2.222222	-1.111111	-2.222222	-1.111111	-0.109089	-11.111111	0.0	-1.111111	-3.456790	-1.111111
2	-0.290553	2.075452	0.609937	3.182479	15.101387	-6.070370	0.0	1.590846	-0.730270	1.347036
3	-2.222222	-1.111111	-2.222069	-1.111111	-1.036443	-11.111111	0.0	-1.111111	-3.407966	-1.111111
4	-2.222222	-1.111111	-2.222222	-1.111111	-11.111111	-11.111111	0.0	-1.111111	-3.456790	-1.111111

Inference:

It's evident that the performance of SMDP Q-learning is not good, as indicated by the reward plot failing to converge over the course of 2000 episodes.

Intra-Option Q-Learning:

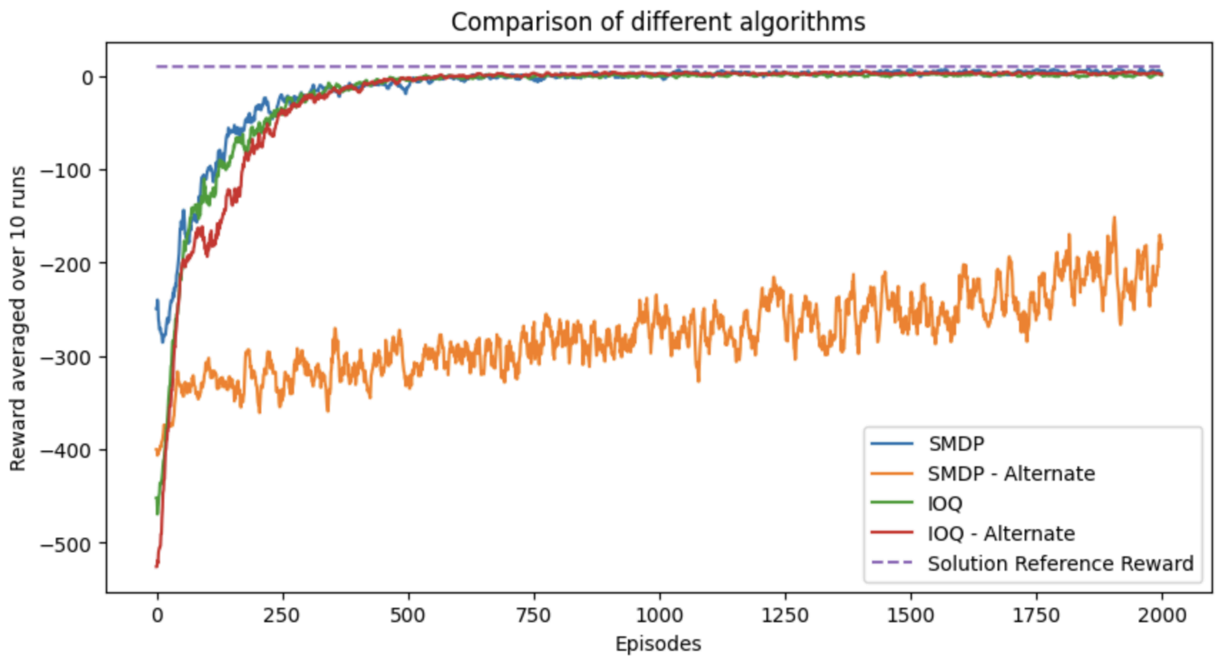




	South	North	East	West	Pick	Drop	Opt_South	Opt_North	Opt_East	Opt_West
0	-0.433861	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-0.877943	-0.763623	-0.820411
1	-2.624176	-0.899020	-2.567441	-0.823201	1.627426	-7.619809	-0.899020	-3.764799	-4.610715	-3.398094
2	0.310301	4.523179	0.917409	2.304070	7.714700	-3.434299	4.523179	0.018272	-2.121616	-1.024384
3	-2.256710	0.348576	-1.734330	-0.564200	2.914016	-6.791747	0.348576	-3.002261	-3.601451	-2.817936
4	-5.812933	-6.163263	-6.034811	-6.133503	-8.836989	-8.705395	-6.163263	-6.792985	-6.632731	-6.719344

Inference:
Unlike SMDP-Q-learning, Intra-Option Q-learning gives better performance even with an alternate set of options.

Comparison between SMDP Q-Learning & Intra-Option Q-Learning:



INFERENCE:
It's evident that, overall, alternate options tend to underperform, particularly in SMDP Q-Learning. This is likely because these options aren't optimized for reaching high-reward states or key objectives like locating, picking up, and dropping off passengers.

Intra-option Q-Learning demonstrates better performance even with these alternative options, nearing convergence with the optimal solution. This improvement is attributed to continuous updates within options during the execution of other options, facilitating faster convergence.

Additionally, the deterministic nature of policies for alternate options contributes, as their state-action pairs are repetitive, leading to frequent updates in the Q-table, aligning with the principles outlined in Theorem 1 of Intra-Option Learning about Temporally Abstract Actions. Theorem 1 (Convergence of intra-option Q-learning) For any set of deterministic Markov options, one-step intra-option Q-learning converges w.p.1 to the optimal Q-values, for every option regardless of what options are executed during learning provided every primitive action gets executed in every state infinitely often.

Both SMDP Q-Learning and Intra-option Q-Learning perform similarly in terms of reward curves and Q-tables. However, the Intra-Option method exhibits superior performance in terms of Q-values and convergence to the optimal policy, as illustrated by the case of state 0.

References:

Sutton, R.S., Precup, D. and Singh, S., 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. Artificial intelligence, 112(1-2), pp.181-211.

Sutton, R.S., Precup, D. and Singh, S., 1998, July. Intra-Option Learning about Temporally Abstract Actions. In ICML (Vol. 98, pp. 556-564).