Programming Assignment 3: Hierarchical Reinforcement Learning

Suraj Rathi
ME19B177

Mrityunjay Upadhyay
OE22S002

Jan-May 2023

---

1 SMDP Q-Learning

1.1 Description

We implemented SMDP Q-Learning for the *Gymnasium Gym's Taxi-v3* environment. SMDP (Semi-Markov Decision Process) is an extension of the Markov Decision Process (MDP) that allows for actions to have variable durations, or in other words, actions can span multiple time steps. SMDPs provide a framework for modeling decision-making in domains with temporal abstraction, where high-level actions or options can be defined to reduce the complexity of the decision problem. The Q-values get updated according to the equation:

$$Q_{k+1}(s,o) = Q_k(s,o) + \alpha[r + \gamma \max_{o' \in \mathcal{O}_{s'}} Q_k(s',o') - Q_k(s,o)]$$

1.1.1 Color Options SMDP

- A *Color* Option is one that can use the {Move North, Move East, Move South, Move West} options to reach a square of the required color.
- A *Primitive* Option is one that can do a single action and then immediately terminates.
- The SMDP has a *Color* option available for each of the colors along with one *Primitive* option for 'Pickup Passenger' and 'Drop Passenger'
- With the limited option set available to it, we can now hide away a large amount of the states for the SMDP. We only expose the following
  1. Which color the taxi is currently at or 'other'.
  2. The location of the passenger
  3. The destination

1.1.2 Color Option

This aims to make the taxi reach the square of a specified color.

- The option is assigned a single color.
- *Initiation:* The option can be initiated at any state. There is no reason to choose that option when the taxi is already at that color, but in practice such an exclusion was automatically learned by the policy
- *Policy:* The option contains its own Q-network.
  - The network stores state as just the (taxi_row, taxi_col).
  - The network can select an action from {Move North, Move East, Move South, Move West}.
  - The network receives a reward as follows:

$$\text{reward} = \begin{cases} 10 & \text{if the taxi is at the respective colored square} \\ -1 & \text{otherwise} \end{cases}$$

- *Termination:* the option terminates if the respective colored square is reached.

1.1.3 Primitive Option

This is a dummy wrapper for a single action.

- The option is assigned a single action.
- *Initiation:* The option can be initiated at any state.
- *Policy:* The network can only choose the assigned action.
- *Termination:* The option terminates at every state (i.e. it immediately terminates).

1.1.4 Test Framework

Average reward is often a very satisfactory metric, but here we felt that it is more important to also track the fraction of times the environment is successfully solved. We run the network in evaluation (non-training) mode for 100 iterations with a set seed.

1.2 Implementation

We will focus on the `policy` and `update` functions for the Color Option SMDP and Color option. In the following snippets, if the eval (short for evaluation) options is set, the network does not modify its internal state (i.e. it does not learn). All the SMDP networks and runners run off a single seeded generator, hence the results are completely reproducible.

1.2.1 Color Options SMDP

```python
        def policy(self, state: int, eval=False) -> int:
            if self.option is None:
                low_state = self.to_lower_state(state)

                self.option_id = self.net.policy(low_state, eval=eval)
                self.option = self.options[self.option_id]
                self.option.start()
                self.l_start_state = low_state
                self.r_bar = 0.0
                self.tau = 0

            return self.option.policy(state, eval=eval)
```

If no option is selected, we reduce our state's dimensionality and sample from the Q-network to choose one. We then execute the currently selected option's policy.

```python
        def update(self, state: int, action: int, reward: float, next_state: int, done: bool, eval=Fal
            self.option.update(state, action, reward, next_state, done, eval)

            self.r_bar += (self.gamma ** self.tau) * reward
            self.tau += 1
            if self.option.is_done():
                self.option.finish()
                self.option = None

                l_next_state = self.to_lower_state(next_state)
                self.net.update(
                    self.l_start_state, self.option_id, self.r_bar, l_next_state, done, eval, tau=sel
                )
```

We first execute the option's update function. If the option is complete, we use the q-learning update equation to update our SMDP's policy.

1.2.2 Color Option

```
1          def policy(self, state, eval=False):
2              state = self.to_lower_state(state)
3              action = self.net.policy(state, eval=eval)
4              return self.to_higher_action(action)
5
6
7          def update(self, state, action, reward, next_state, done, eval=False):
8              state = self.to_lower_state(state)
9              action = self.to_lower_action(action)
10             next_state = self.to_lower_state(next_state)
11
12             reward = -1
13             if next_state == self.terminal:
14                 self._done = True
15                 reward = 10
16
17             self.net.update(state, action, reward, next_state, done, eval)
```

The above snippet is quite similar to the Color Options SMDP's code. As before, the main operations are reducing the dimensionality fo the state and actions. We should note the custom reward function on lines 12 to 15.

1.3 Performanance

The network was trained for 3000 episodes with relatively standard hyperparameter values. In the below figure we can see the test results and reward curve for the network. To offer a baseline for reference, we show the results of vanilla Q-Learning as well as Color Options SMDP where the outer policy was set by a human expert instead of learned.

The yellow line indicates the performance of the Color Option SMDP network.

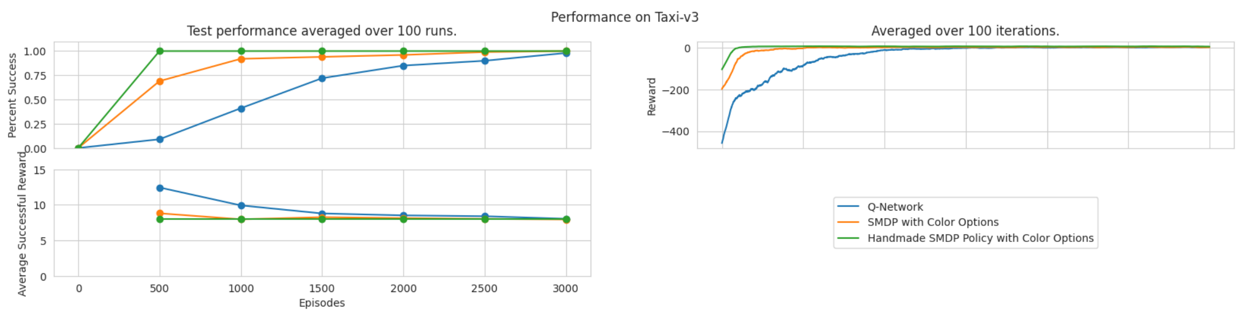


Figure 1: Reward curves and test performance for the Color Options SMDP Q-Learning Network.

We see that the option based framework converges much quicker and shows a higher average reward over the training process. The Q-learning based approach takes at least twice the number of episodes to show similar average reward and 2.5 times the episodes to show similar test success rates.

1.4 Policy Visualization and Description

1.4.1 SMDP Policy

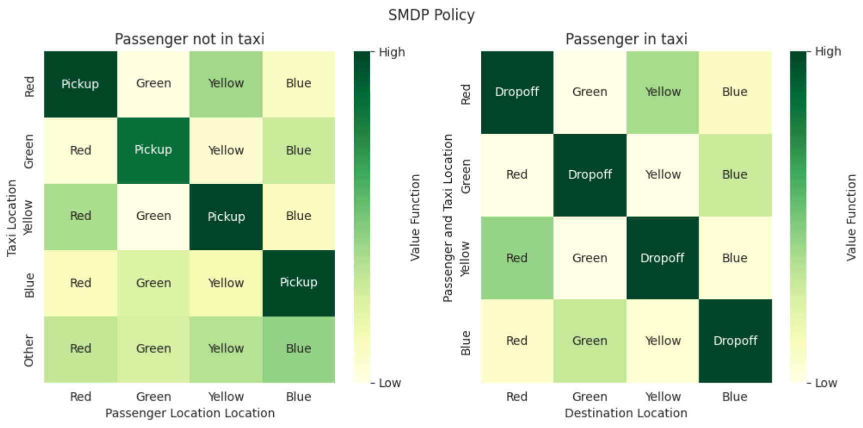


Figure 2: A matrix showing the SMDP's policy and value functions.

In the above figure, we can see a heatmap of the value function of the states in the SMDP's Q-network. The action with the highest value is written in the square. Darker colors indicate higher value. We can summarize the policy in two lines:

1. Go to the color where the passenger is in, and execute the pickup action.
2. Go to the color of the destination, and execute the dropoff action.

This is reasonable. Our goal is to drop the passenger off at the destination in as few steps as possible (reduces the negative reward). The above policy achieves that goal.

1.4.2 Color Option Policy

The goal of the color option policy is to bring the taxi to the required colored square in as few steps as possible. Below we can see the value function for each of the color options as a heatmap and the policy superimposed in the form of arrows.

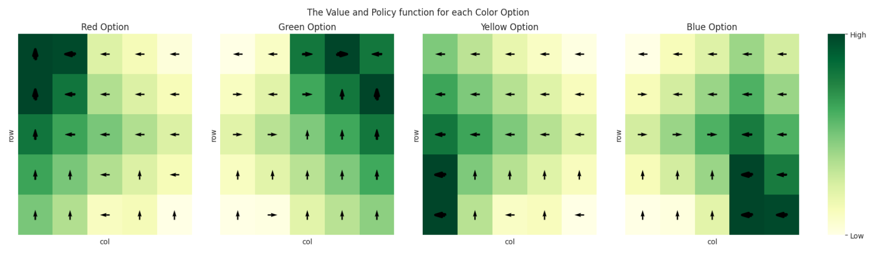


Figure 3: A representation of the policy of the color-options.

The policy is reasonable as it assigns squares closer to the goal color with higher value and the arrows point in the direction of the respective colored square.

1.5 Pick-Drop Option Formulation

We framed a set of options which are mutually exclusive with the given options, we refer to this as the ‘Pick-Drop Formulation‘.

1.5.1 Description

1.5.2 PickDrop SMDP

- The SMDP has the following options available: {Pick, Drop}
- The SMDP has just two states: {Passenger not in Taxi, Passenger in Taxi}

1.5.3 Pick Option

- *Initiation:* The option can be initiated at any state.
- *Policy:* The option contains its own Q-network.
  - The network stores state as just the (taxi_row, taxi_col, passenger_color).
  - The network can select an action from {Move North, Move East, Move South, Move West, Pickup Passenger}.
  - The network receives a reward as follows:

$$\text{reward} = \begin{cases} 10 & \text{if the passenger is in the taxi} \\ -1 & \text{otherwise} \end{cases}$$

- *Termination:* the option terminates when the passenger is in the taxi

1.5.4 Drop Option

- *Initiation:* The option can be initiated at any state.
- *Policy:* The option contains its own Q-network.
  - The network stores state as just the (taxi_row, taxi_col, destination_color).
  - The network can select an action from {Move North, Move East, Move South, Move West, Drop Passenger}.
  - The network receives the same reward as the SMDP
- *Termination:* the option terminates when the passenger is no longer in the taxi

1.5.5 Results

The SMDP learned the following value function. We can see that it tries to pick up the passenger if he isn't currently in the taxi, and tries to drop him off otherwise.

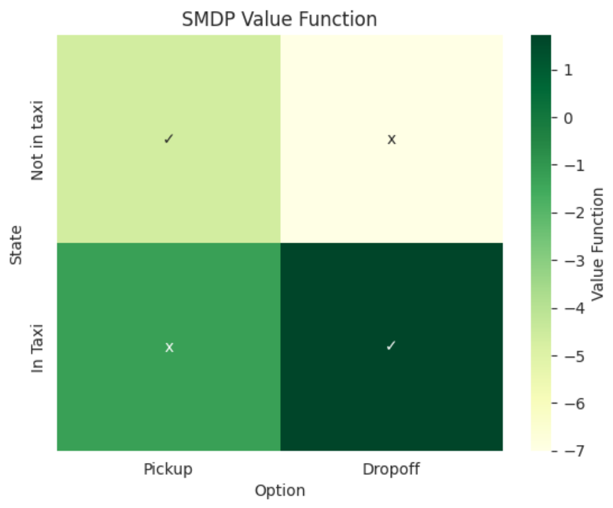


Figure 4: PickDrop SMDP Value Function

We ran three option networks to compare the performance: Primitive Option SMDP, Color Options SMDP, and PickDrop Options SMDP. The Primitive option network performs identically to the vanilla Q-Learning approach. It just wraps each of the basic actions with an immediately terminating option.

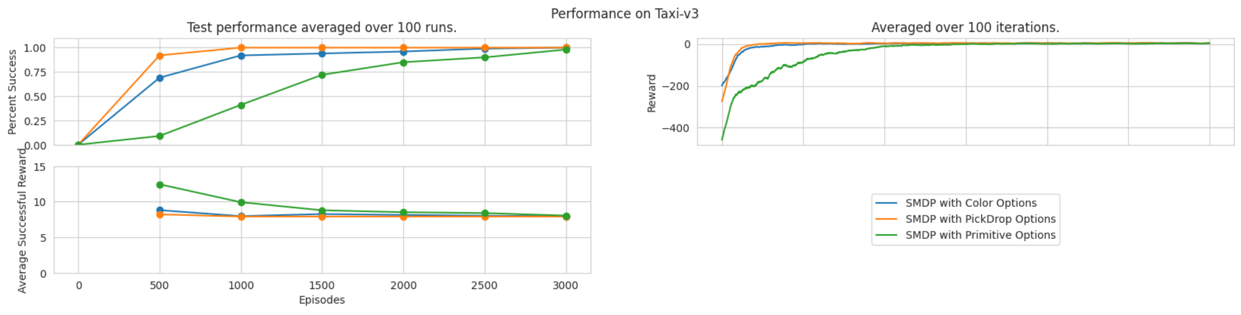


Figure 5: Comparison of different Option formulations.

We can see that the Pick-Drop option formulation performs better than the given (Color) option formulation. It is quicker to converge and shows a higher average reward.

2 Intra-Option Q-Learning

2.1 Description

The main idea behind the Intra-option Q Learning is as follows: "More interesting and potentially more powerful methods are possible by taking advantage of the structure inside each option. In Intra-option learning, whatever be the selected option, the state-action value for the primitive action as well as options that would have selected the same action will be updated." The Q-values get updated according to the equation:

$$Q_{k+1}(s,o) = (1-\alpha)Q_k(s,o) + \alpha[(r_{t+1} + \gamma U_k(s,o)]$$
$$U_k(s,o) = (1-\beta)(Q_k(s,o) + \gamma \max_{o' \in \mathcal{O}_{s'}} Q_k(s,o')$$

For primitive actions (state-action pairs), we use regular Q-learning update.

2.2 Color Option IntraOption , color option and Primitive Option

This has same structure as the SMDP has.

2.2.1 Test Framework

While average reward is typically a useful measure of success, in this case we determined that it was equally important to monitor the percentage of times the environment was successfully resolved. To achieve this, we executed the network in evaluation mode, without any training, for 100 iterations using a predetermined seed.

2.3 Implementation

Our attention is directed towards **policy** and **update** functions for the Color IntraOption and Color option. During the evaluation phase, the network remains static and does not adjust its internal state (i.e., no learning takes place). It is worth noting that all of the IntraOption networks and runners operate using a single seeded generator, ensuring that the outcomes are entirely reproducible.

2.3.1 Color Options IntraOption

```
    def policy(self, state: int, eval=False):
        if self.option is None:
            low_state = self.to_lower_state(state)

            self.option_id = self.net.policy(low_state,  eval=eval)

            self.option = self.options[self.option_id]
            self.option.start()
            self.l_start_state = low_state
            self.r_bar = 0.0

        return self.option.policy(state, eval=eval)
```

The policy method selects and executes an option based on the current state. If no option is selected, it samples an option from a Q-network and executes its policy, while the eval parameter controls if it's during training or evaluation.

```
def update(self,state: int,action: int,reward: float, next_state: int, done: bool, eval=False):
    assert self.option is not None

    self.option.update(state, action, reward, next_state, done, eval)

    self.r_bar +=  reward
    if self.option.is_done():
        self.option.finish()
        self.option = None

        l_next_state = self.to_lower_state(next_state)
        self.net.update(
            self.l_start_state, self.option_id, self.r_bar, l_next_state, done, eval
        )
```

We first execute the option's update function. If the option is complete, we use the q-learning update equation to update our IntraOption policy.

2.4 Performanance

The network was trained for 3000 episodes with relatively standard hyperparameter values. In the below figure we can see the test results and reward curve for the network. To offer a baseline for reference, we show the results of vanilla Q-Learning as well as Color Options IntraOption where the outer policy was set by a human expert instead of learned.

The yellow line indicates the performance of the Color Option IntraOption network.

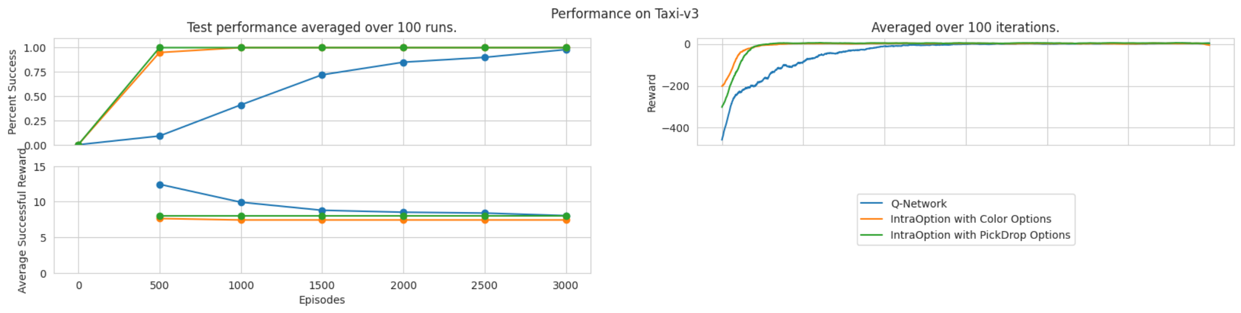
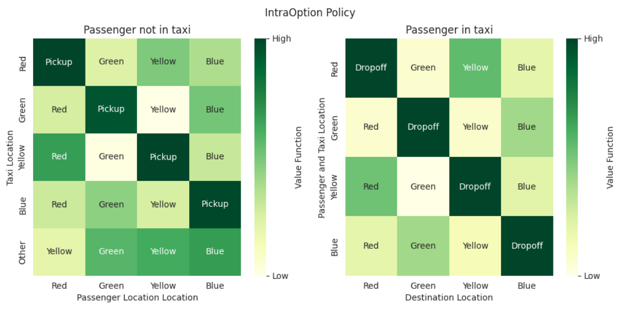


Figure 6: Reward curves and test performance for the Color Options IntraOption Q-Learning Network.

We see that the option based framework converges much quicker and shows a higher average reward over the training process. The Q-learning based approach takes at least twice the number of episodes to show similar average reward and 3 times the episodes to show similar test success rates.

2.5 Policy Visualization and Description

2.5.1 IntraOption Policy



In the above figure, we can see a heatmap of the value function of the states in the IntraOption's Q-network. The action with the highest value is written in the square. Darker colors indicate higher value. We can summarize the policy in two lines:

1. Go to the color where the passenger is in, and execute the pickup action.
2. Go to the color of the destination, and execute the dropoff action.

This is reasonable. Our goal is to drop the passenger off at the destination in as few steps as possible (reduces the negative reward). The above policy achieves that goal.

2.5.2 Color Option Policy

The objective of the color option policy is to minimize the number of steps required for the taxi to reach the required colored square. The heatmap representing the value function for each color option shows that the policy assigns higher values to squares that are closer to the desired color.

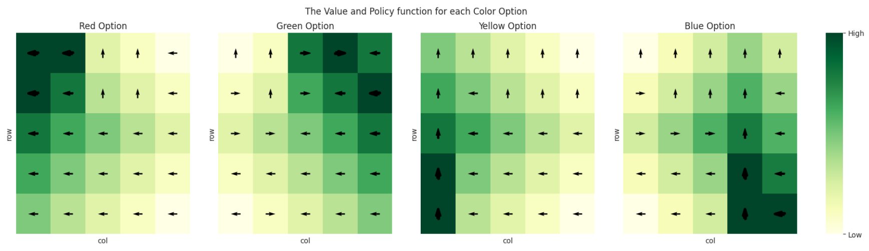


Figure 7: A representation of the policy of the color-options.

The policy is visualized using arrows, which point towards the respective colored square. Therefore, the policy seems reasonable and effective in achieving the goal of bringing the taxi to the required colored square in the shortest possible time.

2.6 Pick-Drop Option Formulation

We framed a set of options which are mutually exclusive with the given options, we refer to this as the ‘Pick-Drop Formulation‘ the definition is same as for SMDP in above section

2.6.1 Results

The IntraOption learned the following value function. We can see that it tries to pick up the passenger if he isn't currently in the taxi, and tries to drop him off otherwise.

2.6.2 IntraOption Policy

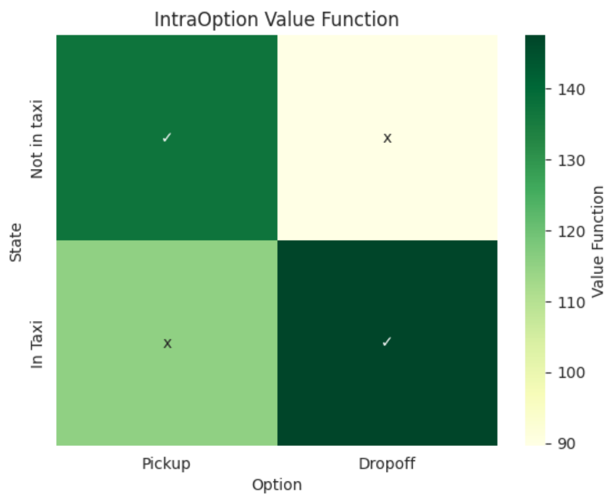


Figure 8: Reward Curve

We ran three option networks to compare the performance: Primitive Option SMDP, Color Options SMDP, and PickDrop Options SMDP and PickDrop Intraoption. The Primitive option network performs identically to the vanilla Q-Learning approach. It just wraps each of the basic actions with an immediately terminating option.

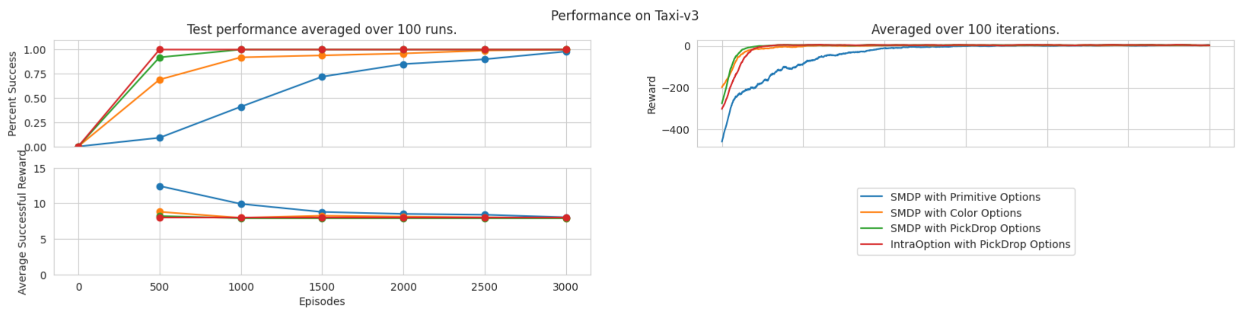
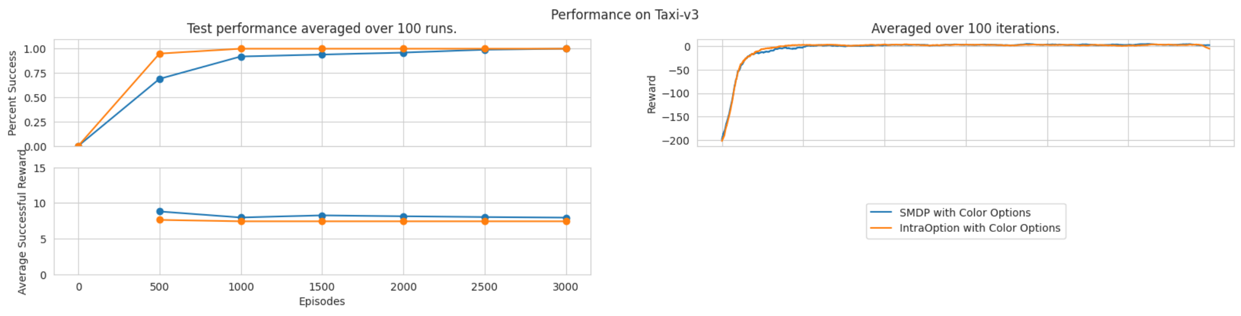


Figure 9: Comparison of different Option formulations.

We can see that the Pick-Drop option formulation performs better than the given (Color) option formulation. And that too for Intraoption rather than SMDP It is quicker to converge and shows a higher average reward.

2.6.3 Comparison between SMDP and IntraOption based on color option

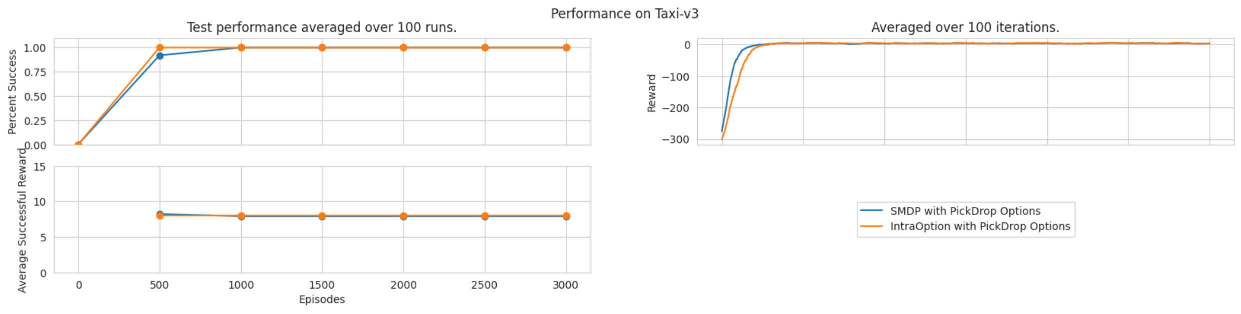


Based on the observation, the results suggest that the Intra-Option with Color Option approach performs better in terms of success rate than the SMDP approach with intra-option. This means that the Intra-Option with Color Option approach is more likely to successfully pick up and drop off the passenger at the desired locations.

However, when it comes to test performance, the SMDP approach with Color Option performed better in terms of average successful reward per episode. This means that when the taxi successfully completed a task, it received a higher reward in the SMDP approach with Color Option.

On the other hand, the Intra-Option with Color Option approach had a higher reward per episode overall, which means that the taxi was able to accumulate more rewards even when it was not successful in completing a task.

2.6.4 Comparison between SMDP and IntraOption based on Pick/Drop option



The comparison between Intra Option and SMDP with the Pick and Drop option in the taxi environment shows that the Intra Option method has a higher success rate in completing the task of dropping off the passenger at the correct destination compared to SMDP over 100 iterations. On the other hand, SMDP with Pick and Drop has a higher average successful reward per episode during the testing phase compared to Intra Option, also averaged over 100 iterations.

Additionally, during the training phase, the reward per episode is higher in the Intra Option with Pick and Drop compared to SMDP with Pick and Drop. Therefore it can be concluded that Intra Option with Pick and Drop is more successful in completing the task but SMDP with Pick and Drop is able to achieve a higher average reward per episode during testing.