Reinforcement Learning: coursework 1

Eamon Dutta Gupta, 01722077

November 2022

0.1 Introduction

This reinforcement learning task required an agent to traverse a maze environment using 3 learning methods, namely dynamic programming, Monte Carlo and Temporal difference. The maze environment included exploring starts, with one high reward target and 3 other 'punishing squares'. Dynamic programming required full knowledge of the environment, including probabilities of transitions between states, as well as the full reward matrix. This was used as a baseline to test the performance of the agents learning via the latter 2 methods which did not have access to the transition or reward matrix.

0.2 Dynamic Programming (DP)

The value function was evaluated using the value iteration method. This was chosen, instead of the policy iteration, as it performs only a single sweep to evaluate the policy once, and then using the Bellman Optimality Equation (1) as an update rule for the value function.

$$V(s) = max_a \sum_{s \in S} P^a_{ss'}(R^a_{ss'} + \gamma V(s')) \tag{1}$$

where V is the state value function, $P^a_{ss'}$ is the probability of transitioning from state s to s' by taking action a, and $R^a_{ss'}$ is the reward collected from taking action a at state s and moving to state s', γ is the discount factor .

An asynchronous approach was chosen also to reduce the computational cost and backup the value function one state at a time. This was also guaranteed to converge given that all states continued to be visited, which was the case since the states were backed up systematically in order of their index. Another design choice was made to construct the halting conditions for the algorithm. Note that the value function was initalised to 0 for all states. This met the condition that absorbing states should have value 0. This did not use 2 arrays since an asynchronous method was used; the value of a state was saved to variable, the value function for that state updated, then the difference of the 2 values was computed.

$$||V_k - V_{k-1}||_\infty \leq \epsilon \tag{2}$$

The value of ϵ was chosen to be 0.001 to provide a balance between accuracy and computational cost. The graph below demonstrates how the the threshold value of the number of iterations. Observing the value functions of each threshold value showed negligible difference (to 2 dp.) hence the reduction in computational cost was favoured.

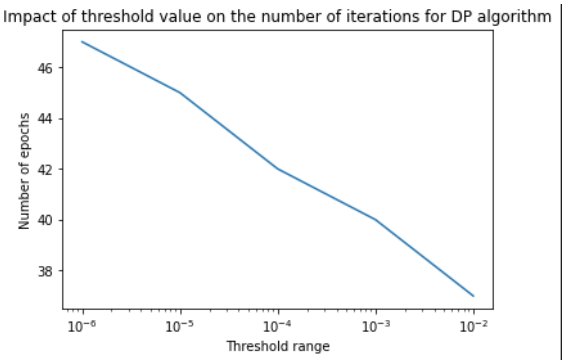

Figure 1: The number of iterations increases with decreasing threshold (presented on a logarithmic scale).

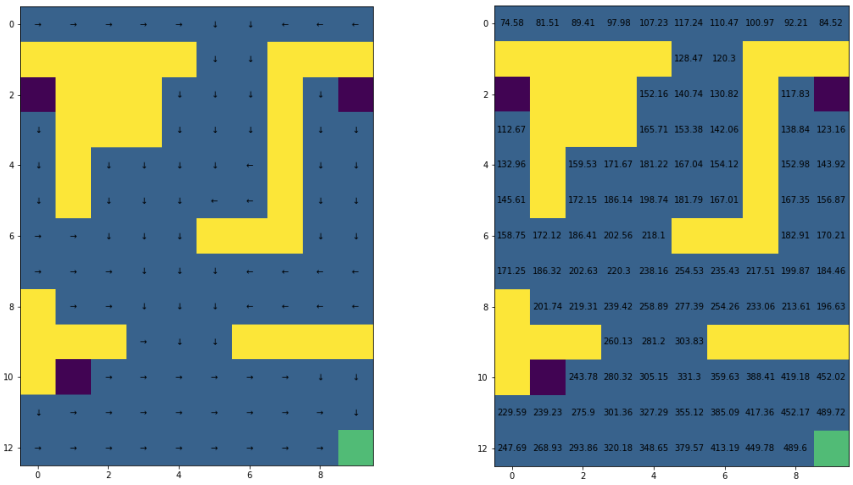

Figure 2: Optimal policy (left) indeed calculates the shortest route to the reward shown in green. The optimal value function (right) shows the expected return of each state following the optimal policy.

Above are the computed optimal policy and value function respectively. The yellow tiles represent obstacles, the purple tiles have a reward of -50, the blue tiles each have a reward -1, and the green tile has a reward of 500. Close to the green tile, the state value functions do take values close to 500. While the policy is deterministic, the probability of a successful transition was defined as 0.84 in this environment, with the discount factor γ being 0.94. Hence the tiles adjacent to the green tile do not take value 500. In this case, the optimal policy also chooses the shortest path to the the reward, which is evidence of a successful result.

The values of γ and p are heavily influential on the optimal policy and the optimal value functions. An interesting case to look at is when $p = 0.25$. As there are 4 possible actions to take at each state, a value of 0.25 represents a completely random 2 dimensional walk through the maze, regardless of the policy since given any action, a transition to another state is equiprobable. As a result, the optimal policy chooses the first indexed action which in this case is to move up, see figure (3). The value function thus represents the expected returns when traversing the maze in a uniformly random fashion.

Another case to consider is $p < 0.25$, this implies when taking an action, 'up' for example, it is more likely to transition 'down', 'left' or 'right'. Thus the optimal policy points away

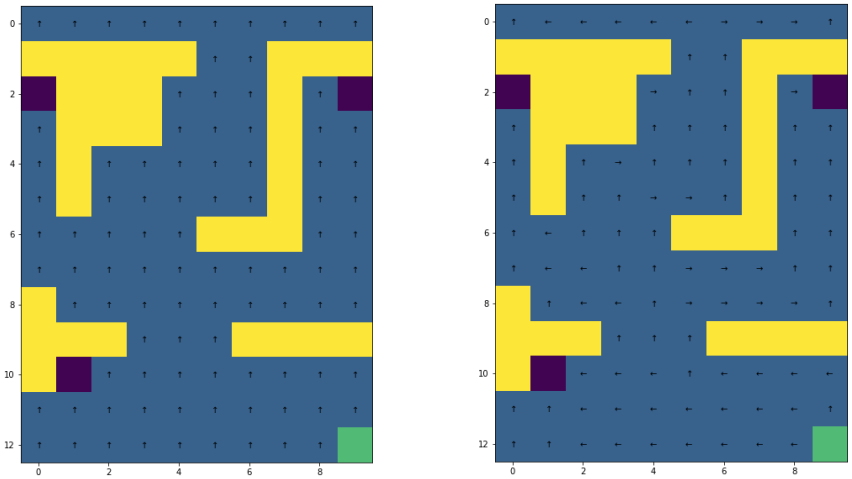

Figure 3: The optimal policies for the DP agent with $p = 0.25$ (left) and $p = 0.1$ (right). Note that the above results were produced by fixing $\gamma = 0.94$

from states with a higher state value. The case where $p > 0.25$ has already been visited in the original setup of the environment where $p = 0.84$, where the optimal policy did point towards states with higher values.

Changing the discount factor to $\gamma = 0.3$ also causes a significant change in the optimal policy and value function. Due to the heavy discount, states in the top half of the maze mostly have very similar state values which are negative. This is because the number of steps it would take to reach the reward from these states means that the heavy discount nullifies the large reward, and that only short term rewards are considered. States in the lower half continue to point in the direction of the reward, as the the number of steps away from the reward is not too large.

0.3 Monte Carlo (MC)

This is the first method that does not assume knowledge of the underlying Markov Decision Process (MDP). The design was an every visit batch MC method with a batch size of 1000. A batch approach was used was used in this environment since the MDP is stationary and did not require frequent policy updates. This approach is also more stable since outlier episodes would not affect the value function as much. For similar reasons, a large batch size was found to be optimal. The state-action value function Q was updated incrementally using the a 'state counter' to count the number of visits to each state action pair, this was used since a learning rate would be unnecessary for an environment with a stationary MDP. The state value counter was also implemented as a hash map to save computational cost. The every visit MC was used as it is more data efficient; it can update the value function for a state multiple times per episode. An on policy method was used in this case, in the interest of computational complexity. For Monte Carlo methods to converge, we require that exploring starts are used, which is satisfied within the Maze class to some extent, as it the agent randomly starts from any square at the top of the maze. Secondly, we require that states will continue to be visited infinitely often - this is achieved by using an ϵ-greedy policy. In order to be Greedy in the Limit of Infinite Exploration (GLIE), the policy must converge to a greedy policy. The ϵ used in this case is given by the following.

$$\epsilon = min(1, \frac{total\_epsiodes}{\beta(episode + 1)}) \tag{3}$$

The β factor defines the proportion of episodes that should be used for exploration. For example with $\beta = 2$ then for the first half of the episodes, the policy remains equiprobable. This increases the probability that the states not in the shortest path to the goal are visited, increasing the accuracy of the value function evaluation. This method of epsilon decay was inspired by from external literature [KW21]. A good value for the hyperparameter β was chosen to be 8.0 as it provided the optimum balance of exploration and learning. However, note that the policy is only updated every 1000 episodes, so using $5 < \beta < 10$ would have a similar effect as the first 2000 episodes are used for exploration for this range of β. The total number of episodes used for MC was 10000 per run, as this allowed enough time for the agent to learn and calculate a stable value function, while not using excessive computational resources.

The policy and value function for the MC method do show significant discrepancies when compared to the Dynamic Programming agent. The policy is not consistent as it sometimes directs the agent away from the optimal reward. This is particularly pronounced in the top half of the maze, whereas the bottom half of the maze often directs the agent in the correct direction as it is closer to the reward. This also means that the optimal value function for the states in the bottom half are more accurate than the top half, where some states take a negative value.

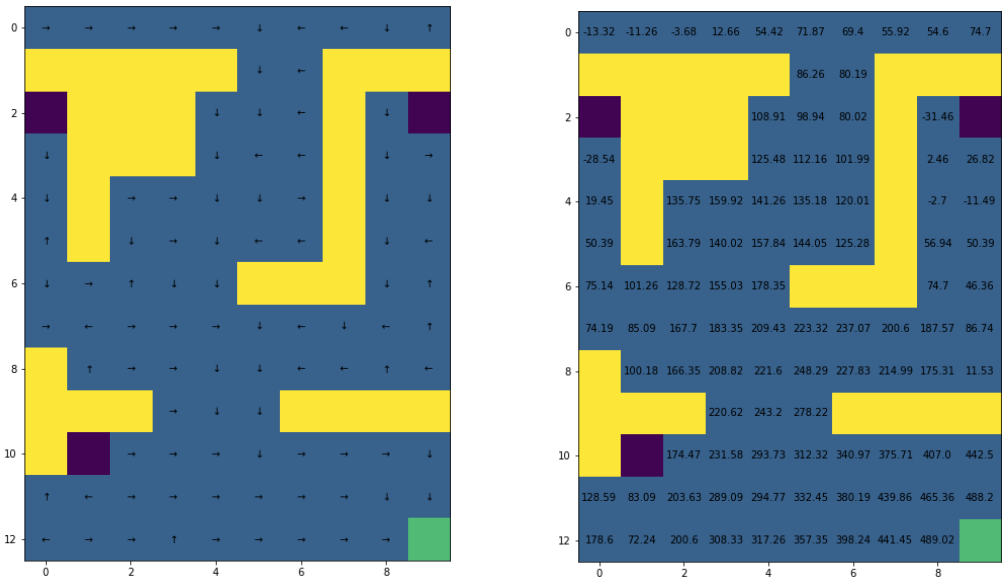

Figure 4: The optimal policy (left) and the optimal value function (right) given by a single replication of the MC agent.

The sources of variability included a non deterministic policy, and uncertainty within the transition matrix, i.e the probability of success is not equal to 1. These 2 factors combined lead to varied exploration phases which in turn lead to varying state values, especially for the states not directly in the path to the reward. In order to decide how many replications to perform for the MC agent, a rolling mean for the value function was calculated, followed by a moving standard deviation of this quantity. In a sample of 50 replications, it was found that after replication number 30, that the moving standard deviation of the rolling mean for every state was less than 3. A threshold of 3 is used as it shows a stabilising mean for the value of that state.

The learning curve for the MC agent shows that for the first 2000 episodes (the exploration phase) the total sum of rewards remains largely negative as the agent finds it difficult to traverse the maze to find the reward in bottom corner. This is due to an equiprobable policy being used for the exploration phase. There is a sudden increase after this phase where the agent follows the epsilon greedy policy and continues to increase until around episode 3000 where it stabilises. The standard deviation for the learning curve is relatively high throughout the

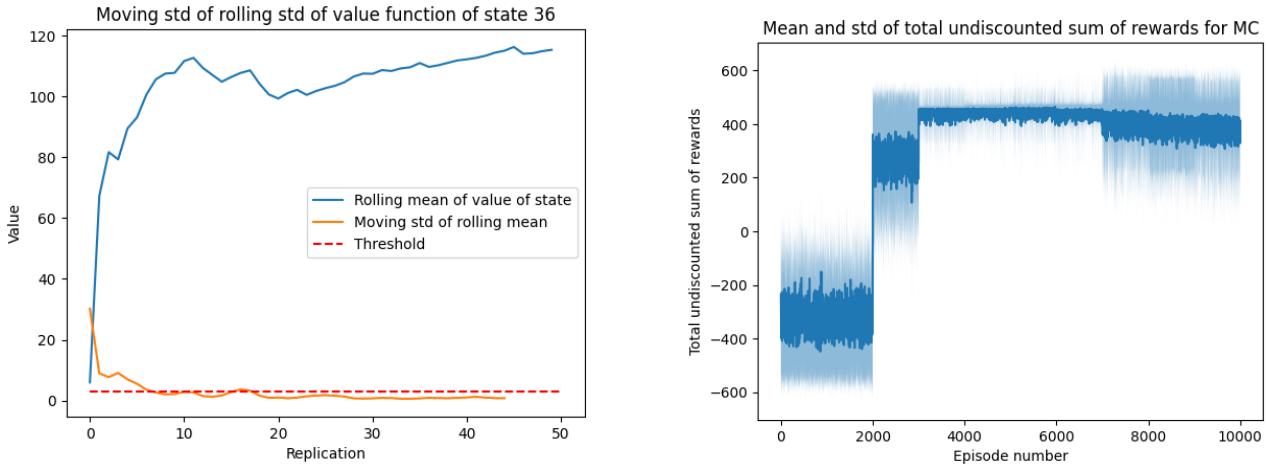

Figure 5: A plot showing how the moving standard deviation of the rolling mean changes with number of replications. On the right, the learning curve for the MC agent for 30 replications.

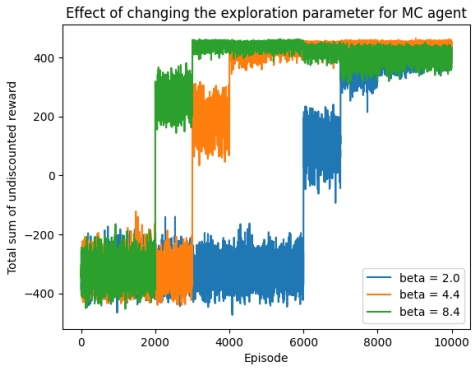

Figure 6: Decreasing the beta parameter for the MC agent results in delayed learning

episodes, but between episodes 3000 and 7000 the total sum of undiscounted rewards reaches a maximum with very small standard deviation. Surprisingly, the performance slightly drops after this, and the standard deviation increases. The explanation for this behaviour is unclear, this may be due to the MC agent being able to travel further down in the maze and beginning to explore other states.

Varying the exploration parameter ϵ via β shows the differences of times when the agent can commence the learning process rather than exploring. As mentioned previously, a larger value of β means that a smaller proportion of episodes will be used for exploration using the epsilon greedy policy defined by

$$\pi(s,a) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A(s)|} & \text{if } a = argmax_a(Q(s,a)) \\ \frac{\epsilon}{|A(s)|} & \text{else} \end{cases}$$

It is known that maintaining sufficient exploration in MC approaches can be difficult, and the epsilon defined in this approach allows control over a fixed proportion of the episodes to be used for exploration. Insufficient exploration would lead to undiscovered states which may provide a shorter path to the reward. Other decaying schemes such as $\epsilon_k = \frac{1}{k}$ were found not be behave as optimally when minimising the Mean Square Error with respect to the result found by the DP agent.

0.4 Temporal Difference (TD)

The final agent used the temporal difference method, also assuming no knowledge of the underlying MDP. The algorithm used was Q-learning, to implement off-policy TD control [SB18]. This was chosen, instead of SARSA due to the convergence speed [HKU16] and lower MSE value using the same parameters. Q-learning also updates the Q value with respect to the target policy which is always at least as good as the current policy. The behavioural policy was initialised as an equiprobable policy and the Q values are initialised to 0 to satisfy the condition for absorbing states. In this case, the same policy Generalised Policy iteration algorithm is used to update the policy in an epsilon greedy fashion, however there is a difference in the policy evaluation step. Similar to the MC agent the ϵ is defined as in equation (3) where $\beta = 2.8$ to allow the agent to allow for an initial exploration phase and satisfy the GLIE criteria. The number of episodes used was 5000, as the algorithm converges quickly (shown in later plots).

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_{t+1} + \gamma max_a Q(s',a) - Q(s,a)) \tag{4}$$

The α parameter characterises the amount by which the current state-action value should shift. The policy is then updated online using an ϵ-greedy approach, in order to meet the convergence criteria given by the Robbins-Monro conditions [Rok19].

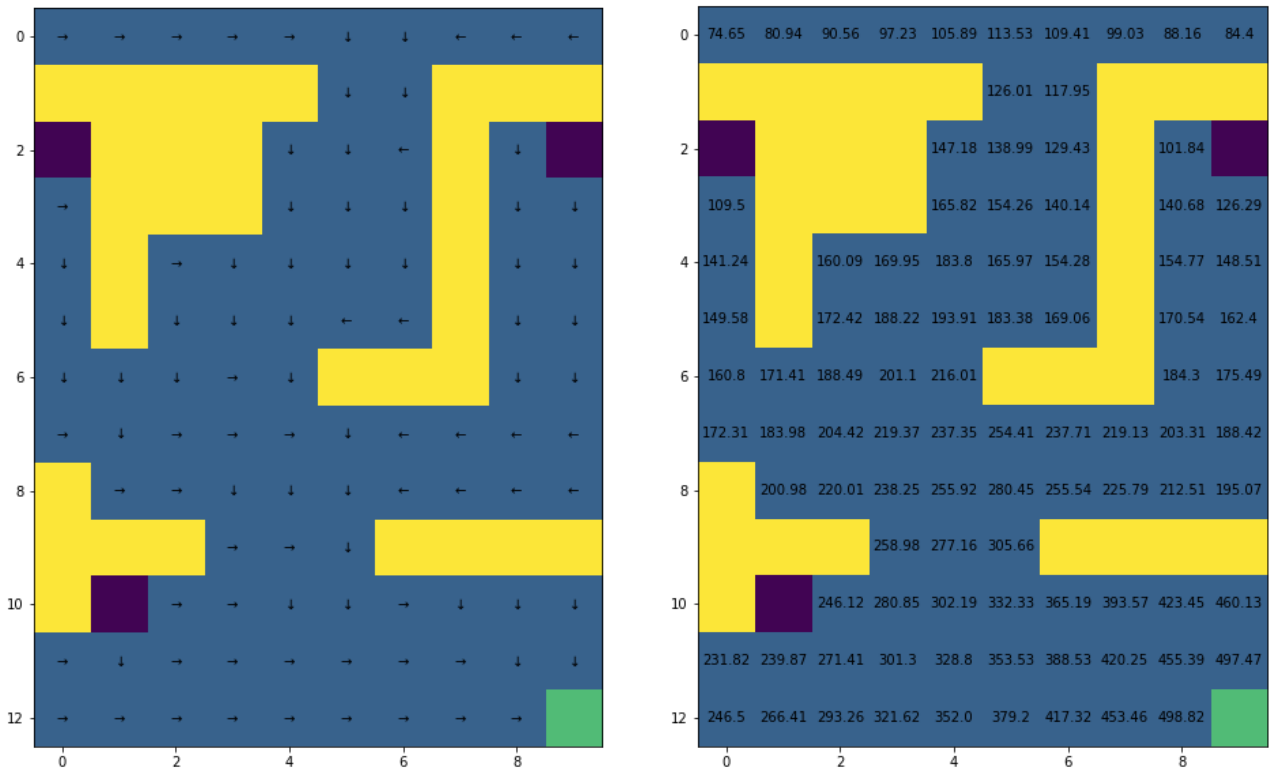

Figure 7: The optimal policy (left) and the optimal value function (right) found by the TD agent in this environment

However it was found that the optimal value for α was a constant value 0.1 which does not satisfy the conditions above. Replacing it with $\alpha_t = \frac{1}{t}$ caused a significant performance drop, hence the α parameter has been fixed.

The Q-learning agent produces much better results than the Monte Carlo method which is evident in both the policy and value function returned. The optimal policy differs from the DP agent only in a handful of states. The value function is also much more comparable to the one obtained by the DP agent. The better performance is since it is not collecting a full trace which is near-stochastic. The Q learning method uses a 1 step trace and then performs bootstrapping to update the Q value. Furthermore, the Q value is updated in the direction according to the target policy which is the optimal one with respect to the Q values, which makes the learning process much faster than the MC agent.

The learning curve plot shows that, as with the MC agent, the total rewards sum is very low for the initial exploration phase where the policy remains equiprobable. After this point at around episode 1700, the agent learns very quickly from the decaying epsilon greedy policy and stabilises at around episode 3000, where the agent continues to get the maximum reward. The standard deviation of the total rewards is high for the exploration phase, as expected, since the agent at this point moves in a stochastic fashion. Then it decreases as the agent follows the optimal policy, with low probability of variation.

Varying the learning rate in figure 8 shows that agents with higher learning rates perform worse with respect to the MSE from the DP value function. Similarly, increasing the beta parameter also resulted in a decrease in performance, as it reduced exploration time. While a decaying learning rate would satisfy the Robbins Munro conditions, a constant learning rate favours convergence as only a finite number of episodes can be run. In this case the learning rate value was not large enough to induce oscillations about the minimum. Out of the tested learning rates, $\alpha = 0.1$ was found to be optimal, along with $\beta = 2.8$. This corresponds to roughly a third of the episodes being used for exploration.

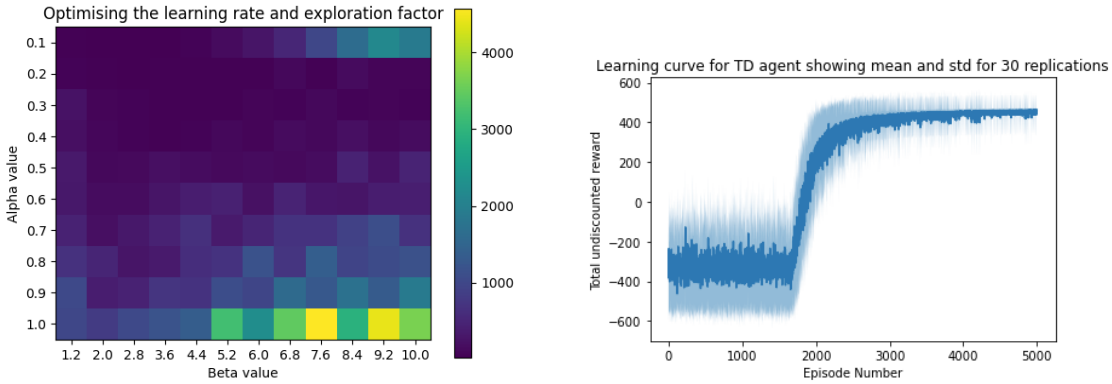

Figure 8: On the left a plot of MSE values compared to the DP value function for combinations of alpha and beta parameters. On the right is the learning curve for the Q learning agent for 30 replications.

0.5 Comparison of learners

The TD agent learns more quickly since it uses an online algorithm, whereas the MC agent uses a batch method. The TD agent is also able to exploit the Markov property present in this environment to improve the efficiency of the prediction evaluation. In comparison, the estimation error for the MC agent only drops after the exploration phase, and plateaus at around episode 6000. Up until this point, the standard deviation for the MC agent is minimal, but then increases after this point, along with the MSE. This correlates with the behaviour seen from the learning curve in figure 5. The MC agent uses full sample traces of the environment and takes longer to learn, with exploration being a necessary factor. The exploration parameter is reduced for the TD agent, as it combines sampling and bootstrapping.

Figure 9 highlights the importance of having a good value function to obtain good rewards. Once the optimal value function has been determined by the TD agent, the behavioural policy is just a soft version of the deterministic target policy. Thus the agent is able to increase the rewards while the MSE remains close to 0. The MC agent on the other hand relies on obtaining the reward to improve its value function, since there is no bootstrapping, and the value of each state action pair is evaluated based on the average return. Thus there is minimal improvement of the value function in the exploratory phase. As the agent learns the location of the reward, it can calculate more accurately the values of states closer to the reward which carry more 'weight' and have a larger impact on the MSE, sharply reducing the estimation error. It can be seen that the TD agent prioritises the value function and therefore is the more efficient agent.

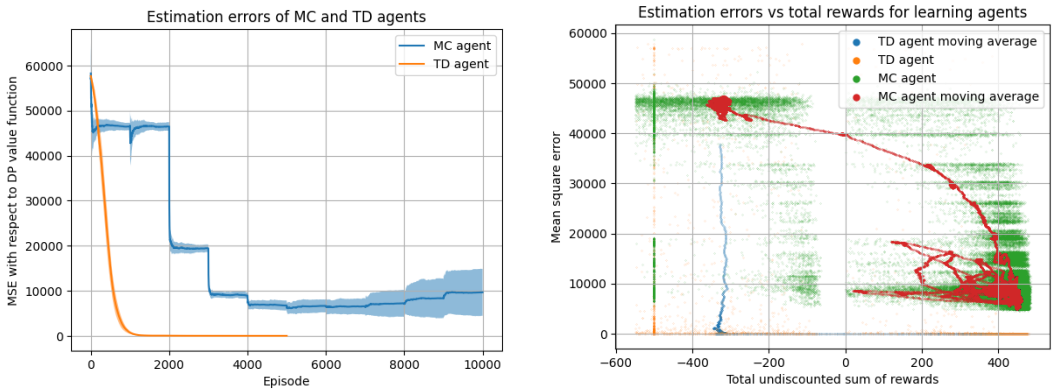

Figure 9: The estimation errors shown for TD and MC agents for 30 replication (left). Scatter plot showing the estimation error against the total undiscounted reward for each agent (right)

Bibliography

[HKU16] Arafat Habib, Muhidul Khan, and Jia Uddin. "Optimal Route Selection in Complex Multi-stage Supply Chain Networks using SARSA()". In: Dec. 2016. DOI: 10.1109/ICCITECHN.2016.7860190.

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[Rok19] Dmitry B. Rokhlin. "Robbins–Monro Conditions for Persistent Exploration Learning Strategies". In: *Modern Methods in Operator Theory and Harmonic Analysis*. Ed. by Alexey Karapetyants, Vladislav Kravchenko, and Elijah Liflyand. Cham: Springer International Publishing, 2019, pp. 237–247. ISBN: 978-3-030-26748-3.

[KW21] Vijay Kumar and Mort Webster. *Importance Sampling based Exploration in Q Learning*. 2021. eprint: arXiv:2107.00602.