Reinforcement Learning

Coursework 1

Department of Bioengineering

Imperial College London

CID: 01701146

November, 2023

1 Dynamic Programming

1.1 Question 1

The environment can be assumed to be a finite Markov Decision Process (MDP), meaning that the future state only depends on the current state and action taken, thus allowing us to build a transition matrix to describe the dynamics of the environment. The environment can also be assumed to be stationary because there is no time-dependent change in the dynamics of the environment and hence we will have a time-independent transition matrix. Since we have only 98 states and 4 actions, we can also assume a small state space and action space.

This problem was solved using an asynchronous value iteration method. Value iteration is the preferred method in this situation over policy iteration due to the following reasons. Given that the state and action space is small and that value iteration effectively combines one sweep of policy evaluation and one sweep of policy improvement in each of its sweeps, the computational requirement of running the value iteration policy is lower than that of policy iteration and hence would converge faster to the optimal policy. An asynchronous method was chosen for this problem because it is more computationally efficient (requiring less memory) and simpler to implement. However, due to the small space and action space, implementing a synchronous method would not have affected the performance of the algorithm too much.

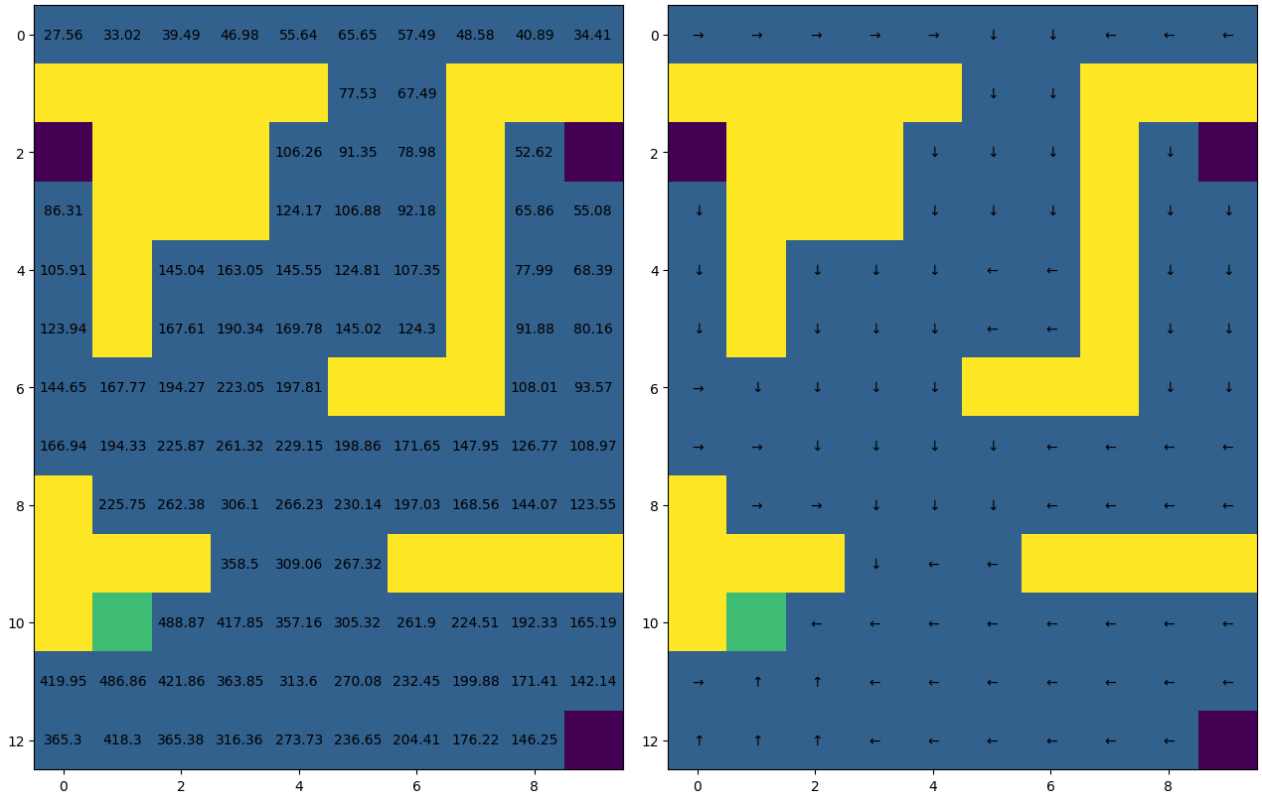
The threshold value (θ) was chosen to be 0.1 because the value function of a state is updated using the Bellman optimality equation:

$$V(s) = max_a(\sum_{s',r} p(s', r|s, a)[r + \gamma V(s')])$$

(where s', r, s, a and V are the next state, reward, current state, action and value function respectively) and since the rewards of the non absorbing states are -1, we require a θ that has a lower order of magnitude than this reward so that the values of each state converge enough to be more representative of its true value and an optimal policy can be extracted. Upon testing lower θ values, I arrived at the same policy, thus we can be confident that the optimal policy was achieved with a θ value of 0.1.

1.2 Question 2

Refer to Figure 1.



(a) Optimal value function

(b) Optimal policy

Figure 1: Optimal value function and optimal policy for Dynamic Programming

1.3 Question 3

The probability (p) determines the probability that the agent will follow in the direction of the chosen action and therefore affects its ability to navigate the maze. The agent moves in the direction of the action with a probability of p and moves in any of the other directions with a probability of $\frac{1-p}{3}$. When $p < 0.25$, the agent is more likely to go in a different direction than suggested by the action. Therefore, we would expect the optimal policy to take actions that do not lead to the goal state in order to force the agent towards the goal state. Since the agent has an equal chance of going into three of the other directions, the environment becomes highly stochastic and we would also expect the optimal value function for every state to be low. When $p = 0.25$, the agent has an equal chance of going in any direction regardless of the action chosen. Since the action does not matter, the optimal policy can have arrows pointing in any direction and the random behaviour would cause low values for the optimal value function. When $p > 0.25$, the agent is more likely to go in the direction suggested by the action. As this value increases, the environment becomes less stochastic and therefore we would expect higher values for the optimal value function and the optimal policy would point towards the goal state (as in Figure 1).

The discount factor (γ) determines how much future rewards are valued. When $\gamma < 0.5$, the agent is more near-sighted and devalues the future rewards. States adjacent to the goal states will have a large positive value because the goal state reward is not discounted as much. As we move away from the goal state, the values of the states decrease sharply until we approach values of just above -1 because the goal state reward is highly discounted and the rewards of moving into another non absorbing state is taken more into account. This makes it hard for the agent to find the goal state and hence, in the optimal policy, the states near the goal state will point in the correct direction but as we move further, they will begin to point in random directions. However, when $\gamma > 0.5$, the agent is more far-sighted and the value of each states have a significant contribution from the goal state because it is not as discounted anymore. The difference in the rewards between the goal state and other states cause the value function of all the states to be positive and they gradually decrease as we move further from the goal state. This allows the agent to identify the location of the goal state and the optimal policy will point towards the goal state.

2 Monte-Carlo Reinforcement Learning

2.1 Question 1

We again assume that the environment is a finite MDP. We assume that the environment is stationary as the dynamics of the environment does not change over time. Additionally, we also assume that the experiences are divided into episodes and all the episodes will terminate regardless of action taken and this is ensured by limiting the maximum number of steps per episode to 500.

The problem was then solved using the on-policy every-visit Monte-Carlo (MC) control (for ϵ-soft policies). The algorithm implements an ϵ-soft policy to navigate the environment and updates the value function and policy according to its experience. The reason this algorithm was chosen over other algorithms like MC prediction and MC exploring starts was to avoid making the unlikely assumption of exploring starts, i.e. the assumption that all state-action pairs will be visited an infinite number of times in the limit of an infinite number of episodes. The ϵ-soft policy satisfies is valid because it continues to allow for exploration and allows the policy to move towards a greedy policy. The on-policy algorithm allows the agent to learn from its own experience to tweak the policy accordingly, and is less computationally intensive. The policy update is carried out in an online manner so that the agent can update its knowledge as soon as it encounters new information and adjust its value function and policy. Furthermore, since the environment is stationary and stochastic, there is a higher chance of the same state being visited again and this extra information from visiting the state can be leveraged by using an every-visit approach to optimise the value function and policy quicker.

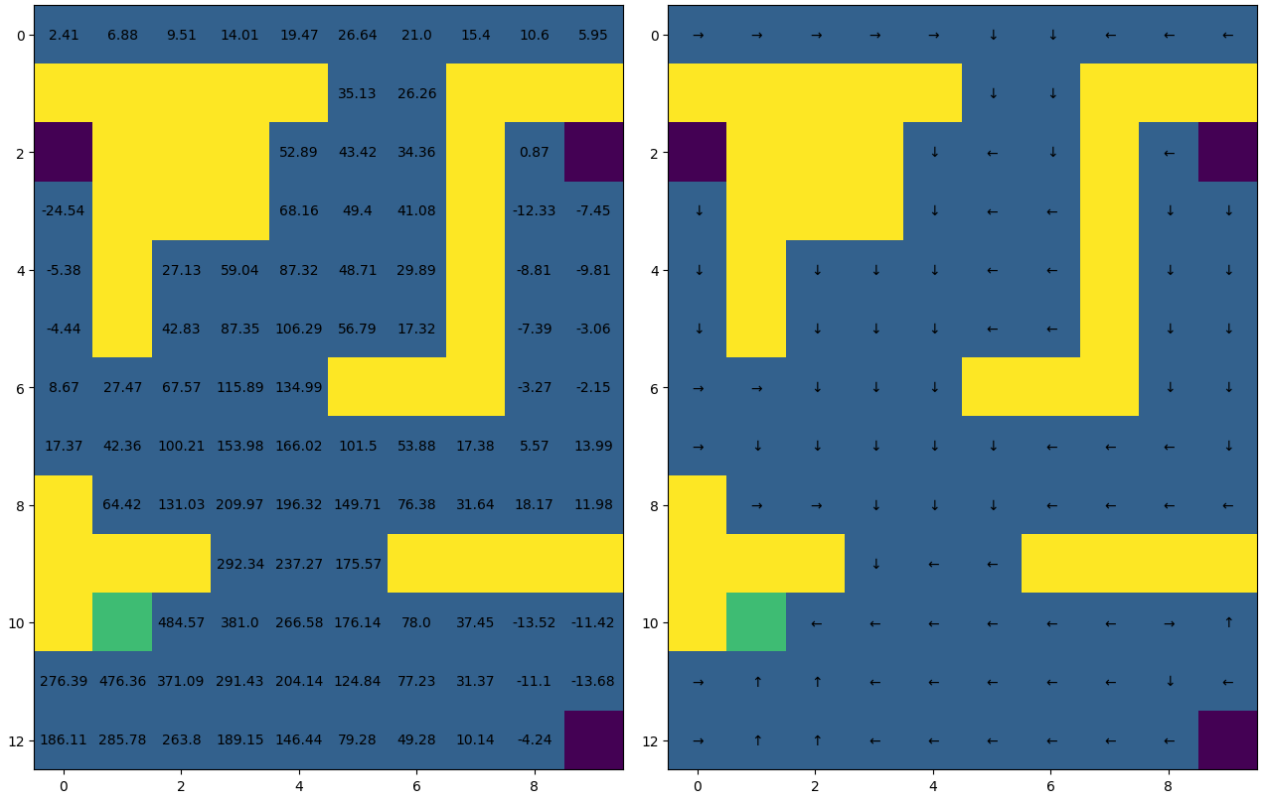
The ϵ-soft policy here is defined as:

$$\pi(a|S_t) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(S_t)|} & \text{if } a = A^* \\ \frac{\epsilon}{|\mathcal{A}(S_t)|} & \text{if } a \neq A^* \end{cases}$$

where π is the policy, a is the action, S_t is the current state, $\mathcal{A}(S_t)$ is the number of possible actions and A^* is the best action at that state. In order to encourage exploration to visit states that would not usually be encountered on the optimal path to the goal state, the ϵ was set to 0.8 and decayed towards exploitation over 5000 episodes using the following function: $\epsilon = 0.9995\epsilon$, where N is the number of episodes experienced. This ensures that states far away from the optimal path to the goal state also have their value function updated to more accurate values, which allows for a more accurate representation of their state policy.

2.2 Question 2

Refer to Figure 2.



(a) Optimal value function (b) Optimal policy

Figure 2: Optimal value function and optimal policy for Monte Carlo

2.3 Question 3

Refer to Figure 3.

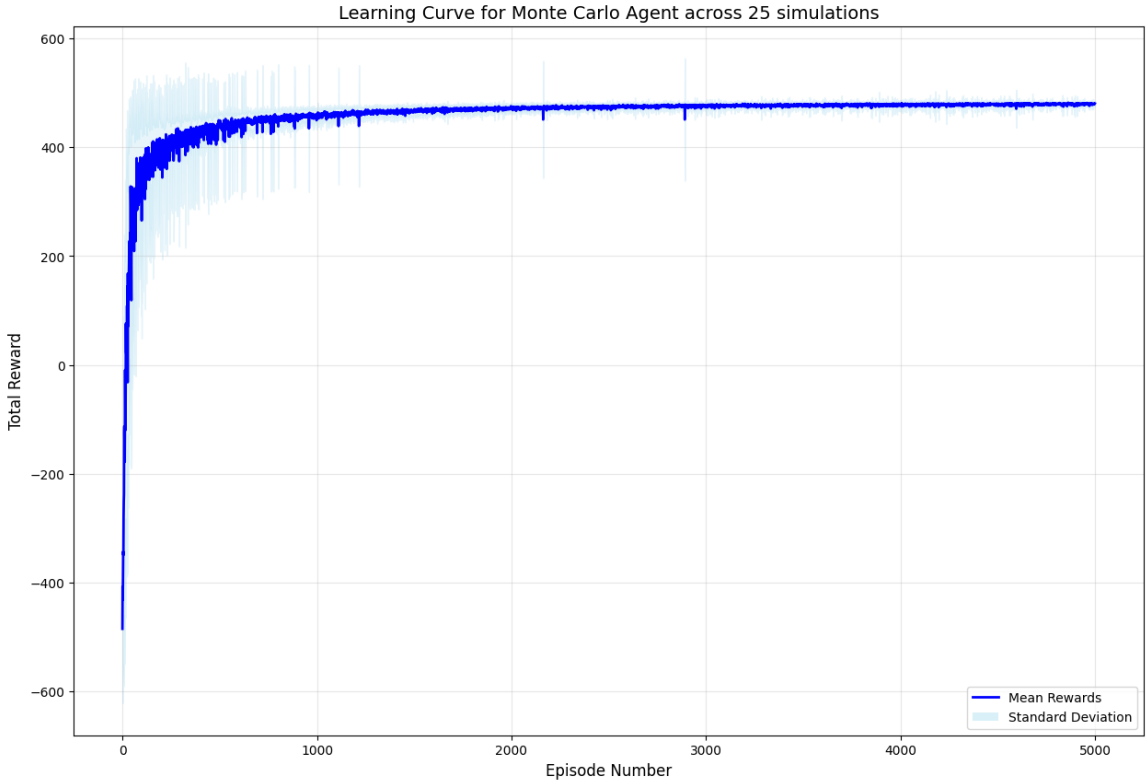


Figure 3: Learning Curve of the MC Agent over 25 simulations

3 Temporal Difference Reinforcement Learning

3.1 Question 1

We again assume that the environment is a finite MDP and that it is stationary. The environment is not very stochastic because the p value is quite high ($p = 0.9$) and hence we can assume that majority of the time, the agent is going to move in the direction of the action taken. This along with the fact that the significant penalty states are far away from the optimal path to the goal state allows us to also assume that the environment is low-risk.

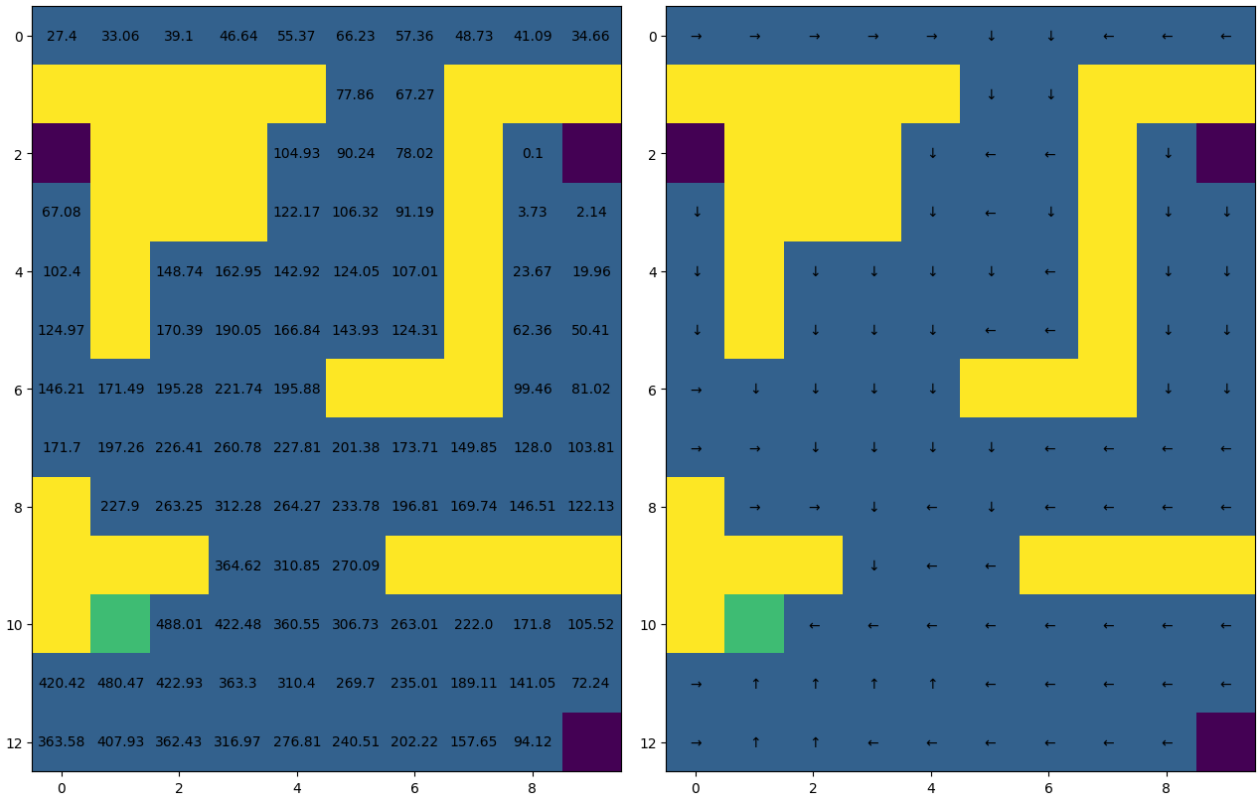
The problem was therefore solved using Q-Learning, a type of off-policy temporal difference (TD) control algorithm where the learned action-value function (Q) directly approximates the optimal action-value function (q^*) whilst being independent of the policy being followed to explore the environment. Q-Learning was preferred over SARSA to solve this problem because it prioritises maximising the expected future returns and therefore quickly finds the optimal path to the goal state whereas SARSA prioritises the safety of its agent and therefore would find a safer path to the goal state. However, since the environment is low-risk, both SARSA and Q-Learning would arrive at the result but Q-Learning would converge faster to the optimal

path because it uses a separate policy to explore the environment and determine the optimal policy.

Q-Learning converges to q^* when a GLIE (Greedy in the Limit of Infinite Exploration) sequence of policies is used and when the Robbins-Munro sequence of step-sizes is used. The first condition is satisfied by using a ϵ-greedy policy that is derived from the learned Q values and ϵ is decayed from an initial value of 0.99 using the following function: $\epsilon = \epsilon(0.9999995^N)$, where N is the number of episodes experienced. A large initial ϵ is used to visit states further away from the optimal path more frequently so as to update their value function to more accurate values and therefore determine the correct policy at those states. The second condition of converging to q^* is satisfied by setting the learning rate (α) to $N^{-0.51}$ (it is valid because $\sum_{t=1}^{\infty} \alpha_t = \infty$) and $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$).

3.2 Question 2

Refer to Figure 4.

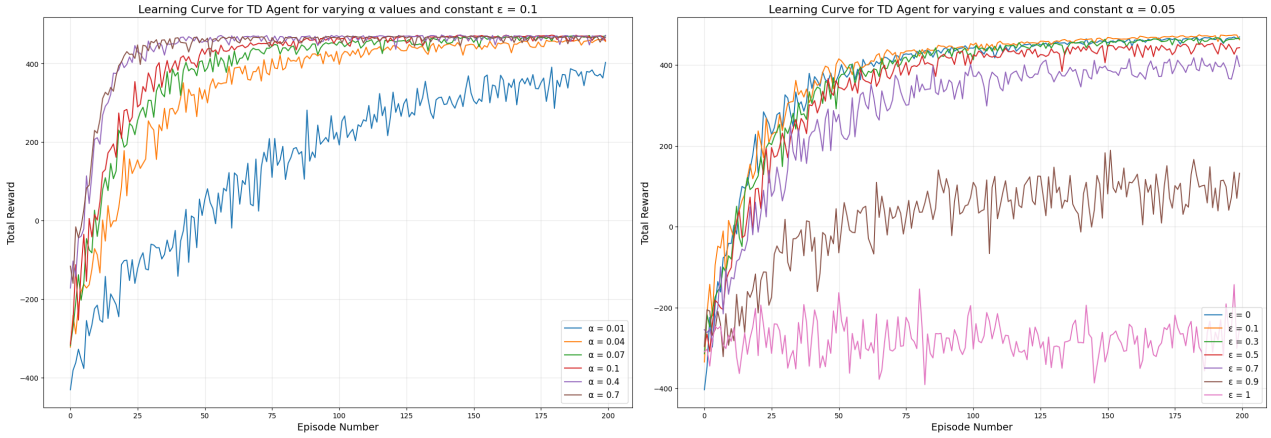


(a) Optimal value function

(b) Optimal policy

Figure 4: Optimal value function and optimal policy for Temporal Difference

3.3 Question 3



(a) Varying α with constant ϵ

(b) Varying ϵ with constant α

Figure 5: Observing the effects of varying α and ϵ

As can be seen in Figure 5(a), the ϵ value was held constant at 0.1 while the α values were varied. By doing so, we can see that when α is small (taking the case of $\alpha = 0.01$), the learning rate is slow and therefore it takes longer to converge to q^*. When α is increased, we can see it begins to converge quicker as well. In fact, when $\alpha = 0.7$, the convergence takes roughly 50 episodes whereas at $\alpha = 0.01$, it continues to converge even after 200 episodes. However, the danger of the fast convergence is that the agent may settle on sub-optimal action-values and therefore never reach the optimal value because the step size of the learning rate is too big.

As can be seen in Figure 5(b), the α value was held constant at 0.05 while the ϵ values were varied. It can be observed that at lower values of ϵ, higher total rewards are collected in a fewer number of episodes as compared to when ϵ increases. This is because ϵ controls the exploitation-exploration balance. When $\epsilon > 0.5$, exploration is favoured and hence the agent travels to many states further away from the goal state, thus accumulating negative rewards and decreasing the total rewards gained. When $\epsilon < 0.5$, exploitation is favoured and the agent prioritises looking for the optimal way to the goal state, thus maximising the total rewards gained. In the special case of $\epsilon = 1$, we can observe that the agent performs a random walk around the environment and the plot does not converge. This is because ϵ dictates the behavioural policy of Q-learning and when $\epsilon = 1$, the agent is equally likely to choose any action at any state. In the inverse case where $\epsilon = 0$, the behavioural policy is deterministic and the agent will always only choose one action, the greedy action which leads it towards the goal state.