CS 747 Assignment 2
Report

Rishabh Dahale
17D070008

October 21, 2020

1 Design Decisions

1.1 Value Iteration

The key idea behind value iteration to solve the Bellman equations is to repeatedly approximate the solution based on current value. Due to this approximation, we can get arbitrarily close to the solution but cant get the exact solution theoretically. Due to this, a termination step is necessary. This termination is generally carried out $V_t^\pi \approx V_{t+1}^\pi$. This condition have been coded up as $||V_t^\pi - V_{t+1}^\pi||_1 \leq \epsilon$. When this condition is met, the algorithm is terminated. Some interesting observations here are as the number of states increase, the terminating condition should impose stricter restrictions on the ϵ because the discount factor can lead to sharper decay of the value function if the rewards are available for few selected states (end state). To incorporate this, I have kept $\epsilon = min\left(10^{-8}, 10^{-\frac{number\ of\ states}{2}}\right)$. Because of this, even if the number of states are less (≤ 64), the condition is strict enough to ensure that the algorithm does not stop before reaching close enough to the real values.

1.2 Linear Programming

While solving the Bellman equations using linear programming, it's necessary to ensure that the solution obtained is optimal. To ensure this, in PuLP library we can check the status of the solution returned. This check is being imposed on line 95 of *planner.py*. So if the MDP supplied to the solver is inconsistent the program will fail this assert and exit and not provide any solution.
I have assumed that the default precision of the solver is valid limit for the MDPs.

1.3 Howard Policy Iteration

For this algorithm, at every time step, we need to calculate the solution of Bellman equations to get V^π. Using these V^π's we find the Improvable Actions and Improvable States. The system of equations given by Bellman equation can be solved using linear algebra. The formulation is as follows:

$$\text{Let } A = \begin{bmatrix} 1-\gamma T(s_0,\pi(s_0),s_0) & -\gamma T(s_0,\pi(s_0),s_1) & \dots & -\gamma T(s_0,\pi(s_0),s_{n-1}) \\ -\gamma T(s_1,\pi(s_1),s_0) & 1-\gamma T(s_1,\pi(s_1),s_1) & \dots & -\gamma T(s_1,\pi(s_1),s_{n-1}) \\ \vdots & \vdots & \ddots & \vdots \\ -\gamma T(s_{n-1},\pi(s_{n-1}),s_0) & -\gamma T(s_{n-1},\pi(s_{n-1}),s_1) & \dots & 1-\gamma T(s_{n-1},\pi(s_{n-1}),s_{n-1}) \end{bmatrix},$$

$$X = \begin{bmatrix} V_0^\pi & V_2^\pi & \dots V_{n-1}^\pi \end{bmatrix}^T \text{ and } B = \begin{bmatrix} \sum_{s' \in S} R(s_0,\pi(s_0),s') \cdot T(s_0,\pi(s_0),s') \\ \sum_{s' \in S} R(s_1,\pi(s_1),s') \cdot T(s_1,\pi(s_1),s') \\ \vdots \\ \sum_{s' \in S} R(s_{n-1},\pi(s_{n-1}),s') \cdot T(s_{n-1},\pi(s_{n-1}),s') \end{bmatrix}$$

We can write the system of linear equations as $AX = B$. To get X we can simply invert A and calculate $A^{-1}B = X$.

But for episodic MDP, gamma can be 1. This can lead to matrix A being non inevitable. This happens because for the end state, the value function should be zero. To handle this case, I have simply made all the transition probabilities and reward zero for the end state. Due to this, the final equation for the end state balances out the coefficients giving us the the required solution correctly.
NOTE: I have assumed that in the MDP file, for continuous task, the end state is given as -1. If this condition is violated i.e. for continuous MDP if by mistake end state is given, then the solution given by the solver can be wrong as the rewards and transition probability are being different.

2 Maze Problem

The maze have been formulated so that if there are more than one end states, the encoder.py will still encode it properly and the solver will give the shortest path from start state to closest end state. The maze have been formulated as follows:

2.1 States

Every empty block of the maze (value = 0, 2, 3) is a state. These states have been numbered from 0 to n-1 from top to bottom in the maze.

2.2 Actions

Every state have a possible of 4 actions: S, E, N, and W. They have been encoded as follows: $S \rightarrow 0$, $E \rightarrow 1$, $N \rightarrow 2$ and $W \rightarrow 3$.

2.3 Reward

For solving the maze, I have taken reward as 10000 for end state only. It's 10000 and not 1 because as the length of the shortest path increase, we need to keep the number large enough so that the loss floating point precision is as small as possible.

2.4 Transition Probability

For every state, I have maintained a list of valid actions. Valid actions are the actions which result in next place in the maze as an empty space and not a wall. Each of these actions are equiprobable. Therefore transition probability can take only these values: $0,\ 1,\ \frac{1}{2},\ \frac{1}{3},\ \frac{1}{4}$.

2.5 γ

Although this is an episodic task, and we can keep $\gamma = 1$ as it will still lead to solvable Bellman equations, I have kept $\gamma - 0.9$. This is because if there are more than one path in the maze, we want the smallest path. Keeping $\gamma = 0.9$ and reward only on the terminal state, we can ensure that the optimal action is the one along the smallest path as the discounted reward will be maximum for this path. For any other path, the discounted reward will always be smaller than that of the smallest path because $\gamma < 1$