CS747-Assignment 2

Yash Khemchandani - 170050025

October 22, 2020

1 Task 1

1.1 Value Iteration

- The value function is initialized to be 0 for all the states.
- If $\gamma \neq 1$, then the value iteration is stopped when $||V_t - V_{t-1}||_\infty < \frac{1-\gamma}{\gamma}\epsilon$ where V_t and V_{t-1} are the value functions at time-steps t and $t-1$ respectively and $\epsilon = 10^{-6}$. The proof of this was done as a problem in a weekly quiz.
- Else if $\gamma = 1$, then the value iteration is stopped when $||V_t - V_{t-1}||_\infty < \epsilon$ where V_t and V_{t-1} are the same as described above and $\epsilon = 10^{-8}$.
- The optimal policy function is obtained by $\pi(s) = \text{argmax}_a Q(s,a)$ where $Q(s,a)$ is the action value function.
- However, for $\gamma = 1$ this still doesn't ensure that $||V_t - V^*||_\infty < 10^{-6}$, where V^* is the optimal value function. In order to get the accurate value function, we recalculate the value function from the policy function by solving the Bellman's equations. This is based on the assumption that the policy function is indeed optimal.

1.2 Linear Programming

- For solving the LP problem, we import `pulp` module in Python and make use of the `PULP_CBC_CMD` solver with the default parameters
- The optimal policy function is calculated from the optimal value function as described in Section 1.1

1.3 Howard's Policy Iteration

- The policy function is initialized to be 0 for all the states.
- After getting the new policy by running the policy improvement, the value function V_π for the given policy is calculated by solving the Bellman's equations. The linear equations are converted to the form $Av = b$ where v is the value function vector.

- If $|A| \neq 0$, v is obtained using `np.linalg.solve(A,b)`, which gives the exact solution for v
- Else, v is obtained using `np.linalg.lstsq(A,b)` which gives the least square solution to the given linear equation

- The iteration is stopped when the new policy function is identical to the old policy function

2 Task 2

2.1 Maze MDP Design Details

- Number of states in the MDP is equal to number of non-wall cells (0,2,3) in the grid.
- From each state, there are a total of 4 actions possible namely − North(0), East(1), South(2), West(3).
- γ for the MDP is taken to be 0.99. Ideally, the value of γ should have been 1 but it was observed that it leads to very slow convergence for large number of states. We also compensate for the diminishing value function by having large positive rewards at the end state as described later
- If an action a from a state s results in an end state s', $T(s,a,s') = 1$ and $R(s,a,s') = \alpha * 10^5$ where T is the transition matrix, R is the reward matrix and $\alpha = \frac{1}{\max(10^{-10}, \gamma^{2(r+c)})}$, where r and c are the number of rows and columns of the grid respectively. We take the exponent of γ to be $2(r+c)$ because on average, the shortest path from start state to end state is assumed to be less than $2(r+c)$ and this ensures that the value of a state is not affected by the diminishing rewards due to the discount factor. And 10^{-10} is added to put a limit to the maximum reward a single transition can have.
- If an action a from a state s results in a collision with the wall, there is a self-loop on the state in the MDP, i.e., $T(s,a,s) = 1$. In this case, $R(s,a,s) = -10^3$ and this higher reward is because of the fact that once there is a collision with the wall, the episode will never lead to the smallest path.
- Otherwise, if an action a takes a state s to state s', $T(s,a,s') = 1$ and $R(s,a,s') = -1$. This negative reward (along with the one above) are used to penalize any deviations from the shortest path.
- The planning algorithm used to solve the MDP is **Value Iteration**. This is because it was observed that this algorithm takes the least amount of time to solve `grid100.txt`

2.2 Proof of Correctness

Let the optimum value function of the generated MDP be V_π, start state be s and the end state be s'. Then the optimum value for the start state is given by:

$$V_\pi(s) = -r_0 - \gamma r_1 - \gamma^2 r_2 - ... - \gamma^{m-1} r_{m-1} + \gamma^m r_{end}$$

where r_i is the absolute reward at time-step i and r_{end} is the reward at the end state. Since V_π is optimum, $V_\pi(s)$ should be maximum, which means that m and r_is should be minimum. This will happen

when the optimum policy takes the shortest path from start state to end state (thus minimizing m) and doesn't collide with the wall at any time (thus minimizing r_is). Thus, the optimal policy function for this MDP corresponds to the shortest path from start state to end state.

2.3 Sample solution for a maze

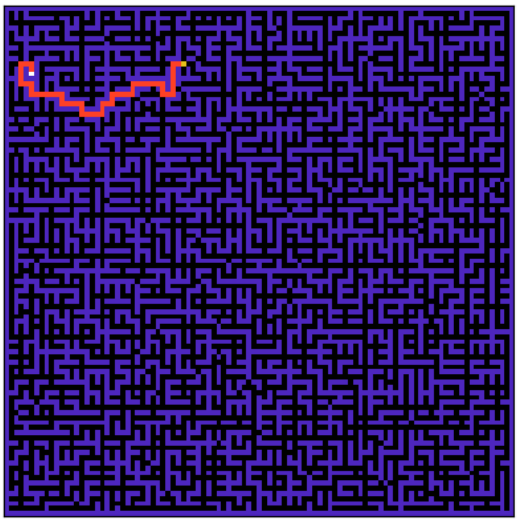

Figure 1: Solution for Maze `grid100.txt`