CS 747 Assignment 2

Teja Bale, 200050020

Autumn 2023

Contents

1 Task1

I used vectorized operations for all three algorithms. Value Iteration is the default algorithm with precision of 1e-9. I constructed two matrices of size (n_a, n_s, n_s) (where n_a is number of actions and n_s is the number of states), named transition and reward, respectively. Here the transition is 3D numpy array which contains transition probability from s1 to s2 given action a1 by using transition[a][s1][s2], similarly reward numpy array contains the reward for the transition described above. I defined a function called get_values which returns the value for each state given a policy and the above transition, reward and discount as the parameters for the function.

1.1 Value Iteration

For value iteration, I have implemented the algorithm discussed in the class and updated the Values until the difference in updated values to previous ones is less a threshold equals to 1e-9

1.2 Howard's Policy Iteration

I initiliazed the policy with some random values. In each iteration, we calculate the action values for all the states, and see if there are any improvable states. I updated the policy for all the improvable states. The action with maximum value is chosen in this implementation. I used the get_values function in each iteration to obtain the values for each state.

1.3 Linear Programming

I used PulP for implementing a Linear Programming based solution. I first defined a set of variables and constraints using PulP and provided the LP with an optimization function. There are a total of $n\_a \times n\_s$ constraints. There is a constraint for every state action pair. I named the constraints as action—state After solving the LP, I looked at the values of all constraints and found the tight constraints by their values, i.e. I consider constraints with values less than 1e-7 to be tight constraints. We get a tight constraint for each state. Based on the constraint name encoding I previously mentioned, I assigned the optimal action for each state based on the tight constraint for that state. In addition, if the MDP is episodic, I added additional constraints to emphasize that all end states have value 0.

Observations: I observed that all of the above techniques were returning the correct optimal policy every for all the test cases. However, a higher value of epsilon(precision value) was needed in the case of value iteration for episodic tasks. (there will some difference for 1e-7 and 1e-9 epsilon values)

2 Task 2

2.1 Encoder

States:

There are a total of 8192 states covering all the possible positions of the $B_1, B_2 \& R$ and possession of the ball, and I created the extra two states for the lost state and winning state. In the input, the states are given in the format [B1 B2 R ball possession], so to give input to the planner I made a dictionary to map these states to 0-indexed indices (string to int) while reading them, and also I created the index to state dictionary (int to string)

Transitions

Instead of creating the entire n_a x n_s x n_s, I have only maintained the transitions where there is a non-zero probability for the transition. For this, I created a dictionary to map string state—action—next_state to probability(float) for the transition.

for each state index, from the given policy of R, I have to find the next position of R, and they have given the probability of R for that action of R. So after updating the R position on the 4 x 4 square, I have iterated to each action (total 10) and calculated the next state for that action and calculated the corresponding probability of the transitions. Here, the transition probability involves the probability of the action of R and the action taken by B1 and B2. So, for every state, we find the possible next states, fill in the transition values, and take care of out of the bounds, which will have the next state as L, and for the successful goal, has state W(which are the last two states)

Rewards

The reward is one only when the next state is the Winning state (which is the last state here), and for all remaining transitions, the reward will be zero.

2.2 Planner

Calculated the optimal policy and optimal values using the planner.py, giving the MDP created by the encoder to it as the input.

2.3 Decoder

After getting the optimal values and actions from the planner.py the states are in the 0-indexed, so we have to convert them back to string form, I again created the index_to_state mapping using the input file, just like I did in the encoder and printed state, optimal_action, optimal_value

3 Plots

Here, I considered the winning probability from a given state to be equal to the value function for that state. Thus, I first generated the optimal policy and then the optimal value for each state. I used those values for obtaining the plots shown below for the state 0509081 by varying p and q

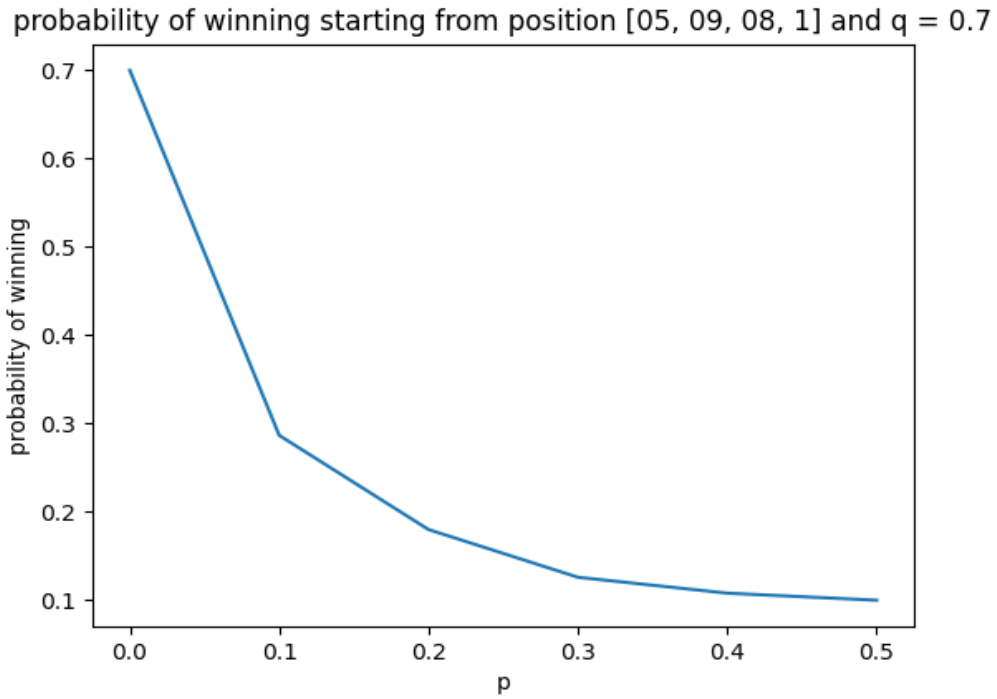

Figure 1: Graph 1

Observations: Here, the value of q is fixed, so the probability of passing the ball and shooting the ball doesn't change. As the p increases, the probability of the successful movement decreases for both players with and without the ball.

So, if the players are far from the goal and try to move towards the goal to maximize the probability of shooting, the chance of failure is 2*p, and it will increase as the p increases and the R tries to move towards the player with ball possession.

So, as the p increases, the probability of scoring the goal decreases.

From the graph, we can see when the p becomes 0.5, if the player moves towards the goal, the player definitely loses since the probability of failure is 1(2*0.5), so to succeed, the player must shoot from that position, and also the R is in the middle, that's why the probability is very low.

When p is zero, the movement towards the goal is 1, so there is a very high chance of getting a goal as the probability of moving towards the goal is high, and the shot probability is high as he becomes nearer to the goal.

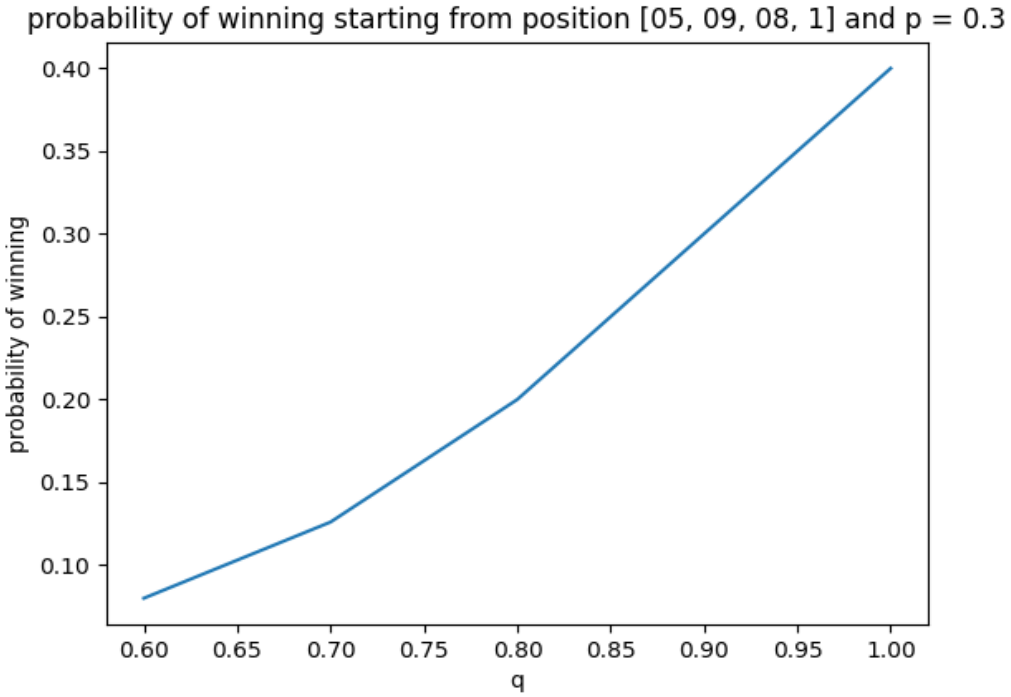

Figure 2: Graph 2

Observations: Here, p is fixed, so the probability of movement is fixed, and when q changes, the probability of goal and passes changes.

From the graph, we can see that as the q increases, the probability of win is increasing. This is because as the q increases, the probability of a successful shoot is increasing(q - 0.2*(3-x1)), which give the increase in the winning probability