CS 747 - Programming Assignment 2

Gowtham S (20D070031)

October 15, 2023

1 Task 1: MDP Planning

After taking in input the MDP from the specified file various parameters like gamma, number of states, number of actions, MDP type (episodic or continuous) are extracted. To store the transitions and their reward, probability, a list named *master_array* is created. This is a multi-dimensional list, which stores only take-able actions and their corresponding transition parameters.

Structure of Master Array:

1. This list contains $|S|$ sub-lists within it. (where $|S|$ is number of states). In case, if the state is an end state, then the sub-list at that index is empty.

2. Each of these sub-lists contains $|A_s|$ 1-D lists, where $|A_s|$ is the number of take-able actions from that state s.

3. Each of these 1-D lists is of the form $[a, (s_1^{'}, r_1, p_1), (s_2^{'}, r_2, p_2), ..., (s_n^{'}, r_n, p_n)]$, where a is the action that 1-D list corresponds to, and each of the tuples after that contain, destination state, reward, transition probability from the given state action pair.

This master array, along with gamma and end-states will be passed as input to the MDP algorithms. Linear Programming algorithm is set to be the default algorithm.

1.1 Value Iteration

Initially, the optimal value function and policy is set to 0 for all states. Then iteratively, these value functions are updated by applying Bellman optimality operator over them. For each state, if it is an end state skip (as it's value is already 0) else we loop through the take-able actions, compute Q(s,a) and update policy and value function which gives maximum $Q(s, a)$.When the $\mathcal{L}^2$ norm of the value function doesn't change by more than 10^{-10}, then the loop is terminated and it is taken to be our optimal value function and policy.

1.2 Linear Programming

In this case, we have $|S|$ variables and $|S||A|$ constraints. The constraints is that the $V(s)$ should be greater than $Q(s, a)$ for all valid state action pairs. It is solved as a minimization problem using **PulP** library. To find the corresponding optimal policy, $Q(s, a)$ is found using the obtained optimal $V(s)$, and that action is assigned to a state which gives maximum $Q(s, a)$.

1.3 Howard's Policy Iteration

We initialize policy for each state with the smallest action index from the set of take-able states from that action. If the set is empty (which is the case for end states), we initialize the action as 0. We then evaluate value function V^{π} for this policy using the custom made $eval\_val\_func()$ function. This evaluation is made based on iteration. Then we loop through each state (skip if end state) and calculate $Q(s, a)$ for every take-able action. We keep a record of the max Q with the corresponding action and if it is found to be greater than the existing value function for that state, then we update the policy for that state. We break the loop if the policy does not change or the $\mathcal{L}^2$ norm of the value function doesn't change by more than 10^{-10}.

2 Task 2: 2v1 Football

The given game can be modelled as an episodic MDP with a discount factor of 1. Since the game is played in 4x4 grid and a state is described by [Player1 position, Player2 position, Defender position, Ball possession indicator], and the game ends if possession is lost or goal is scored,
State Count $|S| = 16 * 16 * 16 * 2 + 2 = 8194$

Action count $|A| = 10$
Discount Factor $\gamma = 1$
There are two end-states, one each for loss and win. The states are zero-indexed. Reward for all transitions are set to zero, except for those, whose destination state is "goal" state. Reward for those transitions are set to 1.

2.1 Encoder

1. Since the states are formatted as string in the input, a *str_to_coord*() function is made to map them to (y,x) coordinates describing positions of players B1, B2 and R. The coordinate system is also zero-indexed, with tile at top left corner as origin, positive x-axis rightwards and positive y-axis downwards. A *coord_to_str*() function is also made to carry out inverse mapping from coordinates, possession combinations to state strings.

2. Probability of transition is calculated as a product of probability of opponent's move and our team move. Opponent move's probability is obtained from opponent policy text file and our move probability is calculated based on the type of action and its corresponding parameters (p or q).

3. While encoding transitions from each state, variables *curr_b*1,*curr_b*2, *curr_r*,*curr_p* and *new_b*1,*new_b*2,*new_r*,*new_p* are created to store positions of players, possessions pre- and post- transition respectively.

4. While passing, since probability varies depending on whether defender is between the players or not, a function *lies_btwn*() is made to check this condition. This is based on the idea that the Euclidean distance between players B1, and B2 must be equal to the sum of the Euclidean distance between B1, R and B2, R.

5. Sometimes, for different movement of opponent, the same state-action of agent, can lead to an end-state. In this case, the probability for that state-action-endstate transitions for different movements of opponent are added over.

2.2 Planner & Decoder

The algorithm used for MDP planning here, by the planner, is Linear Programming. This is just an arbitrary choice, and the other algorithms also perform similarly. The encoder, encodes all transitions in the order as in the opponent policy text file. Hence the job for decoder is quite simple. It takes state strings from opponent policy text file and optimal actions and values from value.txt file rendered by the planner. Since their indices are identically mapped, decoder just prints them together.

2.3 Results & Observations

For the given test cases, it has been found that, the calculated and the given optimal value functions are matching for all states, except for 36, 12, 24 (out of 8192) states for the three test cases respectively.
Given starting state = "0509081"
Since the rewards are all zero except for the "goal" state, expected number of goals scored or alternatively, the probability of winning from a given starting state, against a policy, is nothing but the optimal value function for that starting state under that opponent policy. Therefore, it has been found that,

1. Greedy Defense(p=0.25,q=0.75) : $\mathbb{E}[goal] = 0.175000$
2. Park the Bus(p=0.15,q=0.95) : $\mathbb{E}[goal] = 0.203203$
3. Random Policy(p=0.35,q=0.65) : $\mathbb{E}[goal] = 0.080244$

The following two graphs indicating the probability of winning starting from position [05, 09, 08, 1] (shown below) against policy-1 (greedy defense)

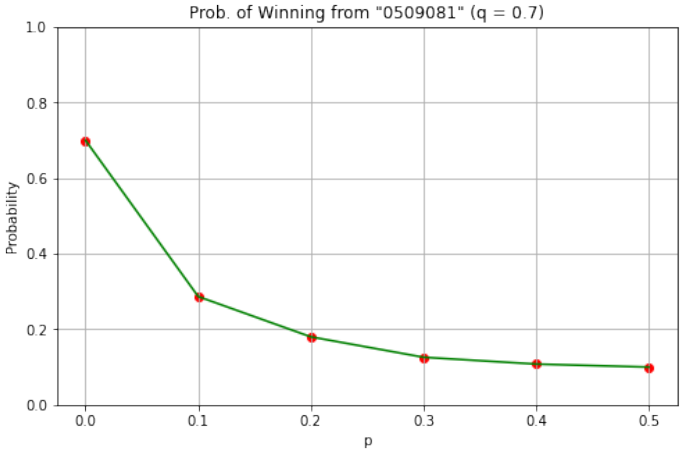

Figure 1: Probability of winning for varying p and fixed q

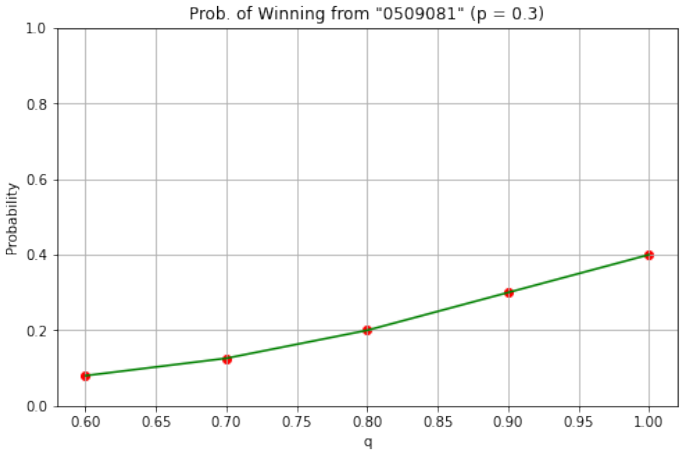

Figure 2: Probability of winning for varying q and fixed p

It can be noticed that,q is an additive parameter in transition probabilities to goal state or other non-ending states. Whereas p is a reductive parameter in transition to non-ending states and also an additive parameter in transition to loss state. Hence, it can be intuitively appreciated that, as p increases, the probability of winning (ie goal scoring) decreases and as q increases, probability of winning increases. In other words, q can be a measure of positive skills of the player, and p can be a measure of his/her weakness.