CS 747 - Programming Assignment 2: Report

Ankit Kumar Misra

October 11, 2021

1 Task 1

Task 1 involved the implementation of three strategies to compute the optimal policy to maximize the reward from any state over a given MDP.

One significant design decision was an effort to reduce the space required for the storage of the given MDP. Instead of storing it in an entire $N \times M \times N$ array, where N is the number of states and M is the number of actions, we store it as two dictionaries for every (state, action) pair. The 'probs' dictionary, in its keys, contains states to which the transition has nonzero probability, and the probabilities themselves in the values. The 'rews' dictionary contains the same states as its keys, and the corresponding rewards in its values.

No assumption as such was made for value iteration; the method was implemented pretty much as it is defined.

For Howard policy iteration, one implementation detail is that, whenever faced with a policy improvement decision, i.e., when there are some actions a such that $Q^\pi(s,a) > V^\pi(s)$, we update the action $\pi(s)$ to be the action a which has the maximum $Q^\pi(s,a)$, whereas the algorithm only specifies that it should be any one of all the potential a's. Furthermore, for the policy evaluation of any policy π, we solve the matrix equation formed by Bellman's equations, and we solve it using numpy.

For linear programming, the PuLP library is used. Equations are framed according to the linear programming formulation discussed in the lectures, and are fed into the PuLP CBC solver. We also remove any zero variables (end states) and zero equations in order to speed up execution. These are added in afterwards.

In all of these, the end states for an episodic MDP have to be treated specially, since they do not have actions rising from them. Whenever we encounter an end state, we simply continue to the next one (as can be seen in parts of VI and HPI), or we remove these states altogether, work with the rest, and add these in at the end (as can be seen in parts of HPI and LP).

We observe that VI is the slowest and LP is the fastest of the three. However, LP does not produce values as accurate as the other two; the last decimal place (when we print upto 6 decimal places) often does not match for the LP solutions. And VI is simply too slow in some cases. Hence, we find a middle way, and we set the default algorithm to be used as HPI.

2 Task 2

Task 2 involved designing an encoder (to form MDP) and a decoder (to interpret value-policy output) for a given fixed adversary in the Anti-Tic-Tac-Toe game.

This is how we design the MDP given the adversarial policy, in the encoder program. Consider a state s for our own player (the states are listed in a file of their own). There are 9 possible actions a (namely $0, 1, \ldots, 8$). We additionally create a terminal state, say t. This is an episodic MDP and t is the only terminal state.

- If a is an illegal action on the state s, i.e., a overwrites one of the spaces that have been previously filled up, then we set a transition with probability 1 and reward -1 from s to t. This prevents any algorithm from taking this route, since losing still has a reward of 0, which is better than a reward of -1 for performing illegal moves.
- If a is an action after which a state s' is reached, and s' is not listed as a valid state for the adversary, that means the game ended with action a. This either means our player completed one of the patterns and lost the game, or the board got filled up and resulted in a draw. Either way, the reward is zero. So, we add a transition from s to t with probability 1 and reward 0.
- If a resulted in a state s' that is listed in the adversary's states, then it is the adversary's turn to play. The adversary has a stochastic policy with various probabilities, each going to a different state. So, for all these states s'' where the adversary could take us with probability p, and which are valid for us, we add a transition from s to s'' in the MDP, with probability p and reward 0. If s'' is invalid, then the adversary either lost or it is a draw. We check the condition on the grid and add a transition from s to the terminal state t with the appropriate reward (1 if we won, 0 if draw), and with probability p.

Note that a reward of 1 is only granted on finally managing to win. Losses and draws have a reward of 0, and any intermediate transitions have a reward of 0 as well.

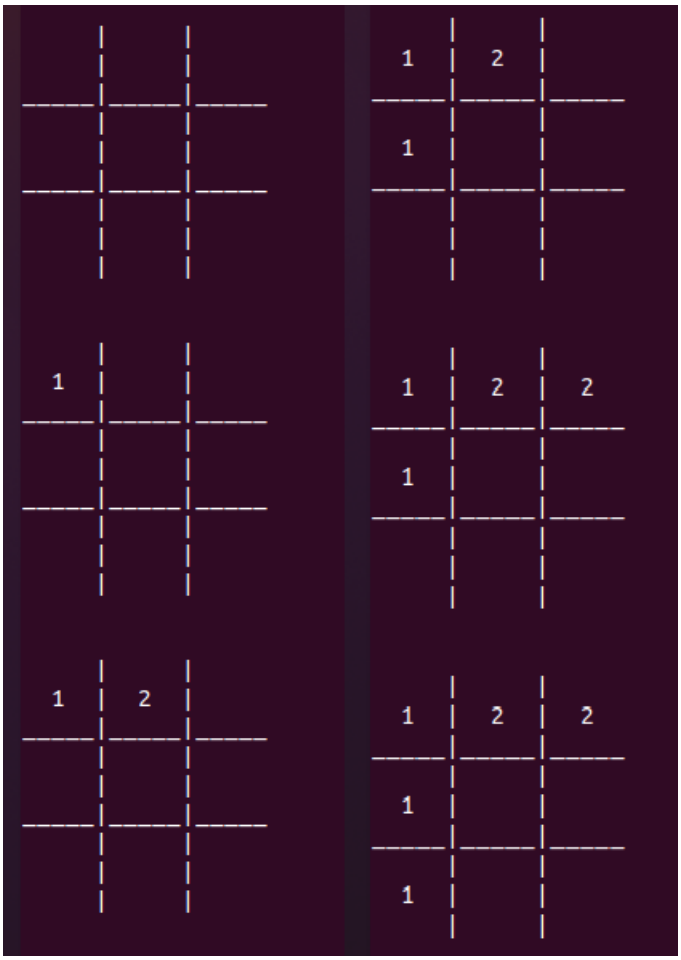
Also note that just doing the above may lead to multiple edges to the terminal state, with the same (s, a, s'') tuple, if some of the adversary's actions cause the game to end. Thus, we have to add up all the probabilities to get the actual probability without suffering from multiple edges. The equivalent reward is appropriately computed as a probability-weighted average of rewards from all the transitions involved, to the end state.

The decoder simply takes the value-policy output from the planner script and expresses it in the policy format, with a probability of 1 for the optimal action at each state, and 0's everywhere else.

3 Task 3

Task 3 involved simulating a competition between the two players, wherein each one strives to get the best possible policy over the other one's current policy, in each iteration.

We observe experimentally that Player 2 always converges to a policy that cannot be beaten by Player 1, no matter what policy Player 1 uses. It is of course possible that Player 1 can draw against Player 2, but since we consider a draw to be equivalent to a loss in our model, we find that, once Player 2 reaches it's perfect strategy (perfect as in Player 2 cannot lose with it; it can only win or draw) Player 1 realizes that it cannot win, and so Player 1 adopts a random losing strategy. No matter whether we initialize Player 1 with a random policy at the beginning, or whether it is Player 2 that has the initial random policy, we discover that, in all the cases that have been computed and observed, the game between the final two strategies (after convergence) goes down as follows:



Clearly, Player 1 is playing horribly and has practically committed suicide. :)

We observe that all the files obtained from the convergence are the same for Player 2. Thus, it seems that this particular strategy is very easy to converge at for Player 2, after only a few iterations, as can be seen from the following (part of) output of `task3.py`.

```
root@800c14645ace:/host/Desktop/CS747/pa2_base# python task3.py
----------------------------------------
Random policy initialized for Player 1
----------------------------------------

Iteration 1

Difference in policies for player 1, in iterations 1 and 2: 434
Difference in policies for player 2, in iterations 1 and 2: 789
Difference in policies for player 1, in iterations 2 and 3: 34
Difference in policies for player 2, in iterations 2 and 3: 124
Difference in policies for player 1, in iterations 3 and 4: 1
Difference in policies for player 2, in iterations 3 and 4: 4
Difference in policies for player 1, in iterations 4 and 5: 0
Difference in policies for player 2, in iterations 4 and 5: 0
Converged!

Iteration 2

Difference in policies for player 1, in iterations 1 and 2: 415
Difference in policies for player 2, in iterations 1 and 2: 800
Difference in policies for player 1, in iterations 2 and 3: 26
Difference in policies for player 2, in iterations 2 and 3: 123
Difference in policies for player 1, in iterations 3 and 4: 1
Difference in policies for player 2, in iterations 3 and 4: 3
Difference in policies for player 1, in iterations 4 and 5: 0
Difference in policies for player 2, in iterations 4 and 5: 0
Converged!

Iteration 3

Difference in policies for player 1, in iterations 1 and 2: 399
Difference in policies for player 2, in iterations 1 and 2: 777
Difference in policies for player 1, in iterations 2 and 3: 37
Difference in policies for player 2, in iterations 2 and 3: 120
Difference in policies for player 1, in iterations 3 and 4: 1
Difference in policies for player 2, in iterations 3 and 4: 5
Difference in policies for player 1, in iterations 4 and 5: 0
Difference in policies for player 2, in iterations 4 and 5: 0
Converged!
```

The program creates a directory called `task3_results` in the same location, with further directories `i1` and `i2` inside. `i1` contains results from iterations that began with a random initialization for the player 1, and `i2` has the results from iterations that randomly initialized player 2. We have 3 such iterations for each of them.

Clearly, convergence is reached in each one of these cases. Also, it is pretty clear that Player 2's converged policy is unbeatable (as in it can be drawed, but one can never win against it), because if there was a policy that had any chance of winning against Player 2's policy, Player 1 would have worked to find it.

Theoretically, we can prove that the alternating program will always reach convergence, as follows:

Proof. The number of policies that can be adopted by any agent in this game are finite, although very large in number, since we are only considering deterministic policies on a finite sized board. The game of Anti-Tic-Tac-Toe has certain policies, for both the first player and the second player, that enable that player to never lose the game; making it either a win or a draw for that player. (Note that if one of the players achieves a policy that can never lose, then the other player can only draw at most, and thus doesn't care anymore because draws are equivalent to losses for us. This is what happened above with player 2 reaching an unbeatable strategy.)

Suppose P1 had a policy a, after which P2 developed b, then P1 developed c and then P2 developed d. Now, d can beat c, c can beat b, b can beat a. Since we are continuously learning, previous policies cannot defeat new policies. So, we go on to better and better policies until either player 1 or player 2 achieves a policy that cannot be defeated. Once this is done, the other player takes up any random policy, and they both stagnate and reach a Nash equilibrium since their environments remain fixed.

The final policy that we converge to depends on various random factors. We observe that P2 usually comes out better, so P2 probably has a higher likelihood of getting to a good policy. □