CS 747: Assignment 2

Pranav Viswanatha 19b090007

October 11, 2021

Task 1

Implemented value iteration, Howard's policy iteration and linear programming approaches to solve MDPs with the default algorithm being Howard's policy iteration. the implementation details are given below. We select a the optimal policy from Q(s,a), calculated as

- **Value Iteration**
  In this we use the Bellman's optimality operators to update values of $V(s)$ for all states.
  $V(s) = \max_{a \epsilon A} \sum_{s' \epsilon S} T(s, a, s')(R(s, a, s') + \gamma V(s'))$
  For tie-breaking, I used the default NumPy argmax for selecting policy from Q, which selects lowest index in case of ties for maximisation.

- **Linear Programming**
  We notice that for calculation of value of V,
  $V(s) \geq \sum_{s' \epsilon S} T(s, a, s')(R(s, a, s') + \gamma V(s')) \, \forall a \epsilon A, s \epsilon S$
  We will try to minimise these sets of linear constraints, as seen in lectures, hence giving us values for V(.) for an optimal policy.

- **Howard's Policy Iteration**
  For this, we start with a random policy, in my case, I used the policy $\pi(s) = 0 \, \forall s \epsilon S$ from here on, for every iteration we calculate V(s) for the policy at the point using the set of $\|S\| \, equations$:
  $V(s) = \sum_{s' \epsilon S} T(s, \pi(s), s')(R(s, \pi(s), s') + \gamma V(s'))$
  We use these equations in the matrix form and we can find V using numpy functions for inverse of matrix and matrix multiplication. From the value of V(s), we determine Q(s,a) and hence update the policy using argmax on function Q. We repeat the process until the old policy matches the new policy.

Task 2

For the encoder, we will use two dictionaries using the states as the key and map them to the line number, for our problem statement player and opp for the opponent player. If we have player 1, then we take number of states in the states file+1 number of states, where the last state is the only terminal state(similar for state 2).

We then look through all possible states and use the policy file to map states to states, and use it to find number of policies with non-zero probabilities and only print those transitions out.

For the decoder, we go through everything and print only corresponding values of policy for each line to be 1 and the rest of them to be 0. This will be the overall policy that the MDP solver selects. For the creation of the MDP, we also add a reward of -1 to the invalid functions, so that it doesn't select an invalid policy.

Task 3

For task 3 we start with an initial value from one of the generated files from task 2 and continue from there. The rewards should theoretically converge under my deterministic tie breaking since even with the case of symmetric outputs, we will have a fixed, lower index tie break. In addition, it is most likely that since the regular tic-tac-toe has fixed policy to always attempt to win or draw, and optimally both players always end up in a draw, it is likely the same will occur here, leaving a 0 reward since both draws and losses are treated the same in our case, it will end up in a not optimal policy again due to selection of lowest index due to constant 0 rewards everywhere. This should create a loop and hence it should not converge to a fixed optimal policy.