CS747: Assignment 2

SUMRIT GUPTA

170040044

Value iteration –

It is solved using Bellman’s equation as given in the slides and code is completely vectorized and no for loop is used in computing the values.

Precision is taken to be 1e-12 for convergence.

The value of the term sum(Reward*transition probability) in bellman equation is constant for a given MDP, hence, it is precomputed outside the while loop for optimization.

Howard Policy iteration –

Policy evaluation is done using the complete vectorized code of value iteration and the stopping condition is when there is no improvable state.

Values V(s) for each state ‘s’ are being computed by solving the linear equations using np.linalg.solve() for each state each time the policy is changed by switching from improvable states.

There can be a case when the complete row of the coefficient matrix is zero and the constant vector of sum(Reward*transition probability) is also zero, which can happen when the probability of transition from some state to all other states except to itself is zero and the reward of transition to itself is zero, the case of episodic MDP in which the end state transit to itself only with reward 0.

This case is handled by setting the term corresponding to the row and column of that end state in the coefficient matrix as 1, which would set the value received from such transitions as 0, i.e. V(endstate) = 0 as suggested in class, without any error of singular matrix.

Linear Programming –

The problem of maximization of sum(-V(s)) for each state ‘s’ is created using pulp along with addition of constraints as mentioned in the slides.

The pulp solver gives the values of V, which is then used to get the optimal policy using Bellman’s equation.

The precision of pulp solver is 1e-5 by default, hence, 6th digit after decimal place may not match with the results.

Formulation of the MDP for the maze problem

I have recursively removed the terminal states (states from where we have only one way to go) from the maze except the start state and end states to reduce the no of states significantly since any path leading to such dead end is not useful. Using this we get only 3155 states for grid100 as compared to 5500+ states if you consider all states except walls.

The possible actions are left(W) = 0 right(E) = 1 up(N) = 2 down(S) = 3

Transitions are completely deterministic, hence T(s,a,s’) is 0 or 1 everywhere. For an action, if the next state is reachable, it's transition probability is 1, else 0.

The encoding is done using dictionary in a hash table from a tile's index in grid to state number so that the path can be traced back by just the value of state number (which can be split easily in ith row and jth column if numbering of states is done in order from 0 to n^2)

The reward for reaching the end states is 10000, and -1 at every other state regardless of the previous state or action taken. The negative reward will penalise any deviations from the shortest path.

The value of gamma is taken as 1 as it is an episodic MDP and gamma = 1 ensures that the values don’t get decimated and the shortest path (path with highest total value as summed over transitions taken from the start state to reach the end state) can be computed accurately even in case of multiple end states.

NOTE: USE VALUE ITERATION FOR MAZE TESTING

Observations about algorithms –

Value iteration is the fastest for maze solving and policy iteration is the fastest for solving MDPs that have large precision on the reward values. Linear programming is the slowest among all since adding constraints to the problem in PuLP is very slow.