CS 747 : Programming Assignment 2

Shalabh Gupta (180050095)

22 October 2020

1 T1 : MDP Solving

Solving Markov Decision problems using Value iteration, Howard policy iteration and Linear programming

1.1 Value Iteration

- Parsing the given mdp files and forming 3 dimensional arrays T and R each of dimensions (states,actions,states). Non transition probabilities and rewards set to 0.
- Criteria for convergence of algorithm has been set to difference in 2-norms (square root of sum of squares) being less than epsilon = 1e-12 (for higher precision).
- After V* has been found, Q* is calculated and π^*(optimal policy) set to argmax of Q* on axis 2.

1.2 Howard Policy Iteration

- T and R as formed in value iteration. Again Non transition probabilites and rewards set to 0. Initial policy pi initialised by choosing a random action for each state.
- Solving **linear bellman equations** for this π to calculate V^π and then Q^π from V^π.
- Finding π^{dash}, actions that would give us highest reward by argmax over Q^π and comparing with π. If $\pi^{dash} = \pi$, none of the states is improvable and algo has converged else change π to π^{dash}.

1.3 Linear Programming

- Converting the problem into linear programming and using **PULP** library to solve. This involves encoding $n * k$ constraints and an objective function to be maximised and solves for V*.
- After V* has been found, Q* is calculated and π^*(optimal policy) set to argmax of Q* on axis 2.

2 T2 : Maze Solving

2.1 Method and Intuition

The problem involves finding shortest path between a start point and one of many end points given in an $n \cdot n$ dimensional maze. We will be referring to the wall in maze as 1, empty cells as 0, start state as 2 and all end states as 3 following the labelling in the problem. The crux of the problem lies in formulating an equivalent MDP solving problem for the given maze. Since we want to minimize the

path, intuitively we should set negative rewards so **that maximising rewards will ultimately lead to short paths** (longer path -> more negative rewards).

2.2 Implementation

- All grid cells except mapped to a state numbering, so total number of states = n*n - [count of 1].
- 4 actions possible at each state, 0 denoting S, 1 denoting N, 2 denoting E and 3 denoting W.
- No assumption on number of end states. Will work even in case of **multiple end states**.
- For each state which is not one of end states, we check the neighbourhood corresponding to the action. If the corresponding neighbour cell is not 1(a reachable grid cell) , we set the transition reward to -1 . While if neighbour cell is 1, we set it to a very low reward($-\infty$).
- For end states, we set all transition probabilities to 0, and transition rewards do not matter now.
- Probability of transition is always set to 1 since given state s and action a, s' is **deterministic** (on going left from x,y you will always reach x-1,y as an example). So T[s,a,s'] = 1
- Discount factor has been set to 1 and problem modelled as an episodic MDP.
- Preferred choice of algorithm is **Value Iteration**.
- The policy thus returned by MDP solver is re interpreted in form of directions from start state to whichever end state is reached first.