# CS747: Assignment 2 Report

Manan Sharma (170040067)

October 23, 2020

## Task 1

This task consists of implementing 3 algorithms (Value Iteration, LP, and Howard's Policy Iteration). The code is modularized, and each algorithm resides in a different file of same name as algorithm, invoked only by *planner.py.* Some algorithm specific details/observations are:

**Value Iteration:** It was carried till the change in each component of the value function vector after an iteration came within a set precision (1e-8 for this assignment). The initial value function was set as 0 for all states. For episodic MDPs, value function for end states isn't updates but set as 0, though this (zero) value is used in RHS of Bellman equations for calculating value functions for other states. For MDPs with higher number of states and actions, value iteration was the quickest to converge of all three.

**LP:** The LP problem was formulated and solved by using Python API of PuLP. The solver is set to return the value function when the consecutive value function (vector) components come within a set precision value of each other (1e-10 for this assignment). The initial value function was set as 0 for all states. Notably, this was the slowest algorithm as number of states and actions were increased.

**Howard's PI:** For each iteration, improvable actions are calculated in each iteration. For each improvable state, the action having highest value for the given policy is chosen, and policy updated. The algorithm only terminates when there are no improving actions for every state (or no improvable states).

For calculating value function (vector) from a given policy in each iteration till convergence, the value function is calculated in a manner similar to value iteration. The inner iteration goes on till the consecutive value function vector's component are within a specified tolerance value (absolutely and relatively).

## Task 2

**Encoding maze as MDP:** In a grid, all positions not having 1, are the states of the underlying MDP. Call these positions, valid positions. So natural mapping of valid grid positions and state number arise by setting state number of a valid position as number of valid positions in the grid that occur in previous rows, and in the previous columns of the valid position under consideration. That is, if [i,j] element of the grid is a valid position, then the state number assigned to it is sum of 1) the total number of valid positions in rows with row number less than i, and 2) number of valid positions in the column segment [i, j'] for all j'<j. Two maps/dictionaries are maintained for mapping a position (i, j) to state number s, and the reverse mapping. Overall, this mapping costs O(N) time to form, where N is total number of elements in the grid.

Now total number of actions are 4 (corresponding to movement to each of the 4 adjacent states). For every action a, at each state s (corresponding to position [i, j] in grid), the value of T(s, a, :) is set as follows:

1) If by taking action a from state s, we reach a valid position in the grid (with corresponding state number s'), then we set T(s, a, s') = 1, and rest values in vector T(s, a, : ) is set 0. (since $\Sigma_{s'}$ T(s, a, s') = 1)
2) Else if the position in grid obtained by taking action a from s is not valid (i.e. a wall is encountered) the taking action a from s takes us back to s, i.e. T(s, a, s) = 1 and for other values T(s, a, : ) = 0

Lastly, for setting rewards; rewards corresponding to all transitions are set as -1 except for the transition to one of the end states, for which the reward is set to a high value (1e8 for this assignment). For all mazes, discount factor is set to 0.9 for all mazes. All mazes are episodic MDPs, but setting discount factor as 1 makes all algorithms to converge very slowly, except perhaps LP for some cases). The over time-complexity for encoding is also O(N).

**Solving the encoded MDP:** The encoded MDP is solved by the previous implemented algorithms as an episodic MDP. The output, as usual, is the value function, and action corresponding to optimal policy, enumerated with increasing state number.

**Decoding the encoded MDP:** Using output from the above solver, the decoder, starting from the start position (state), outputs the action taken for each current state from the optimal policy of the output, and changes the state as per the above action. This continues till any end state is reached.

**Sample output:**

For grid 100 of provided test case:

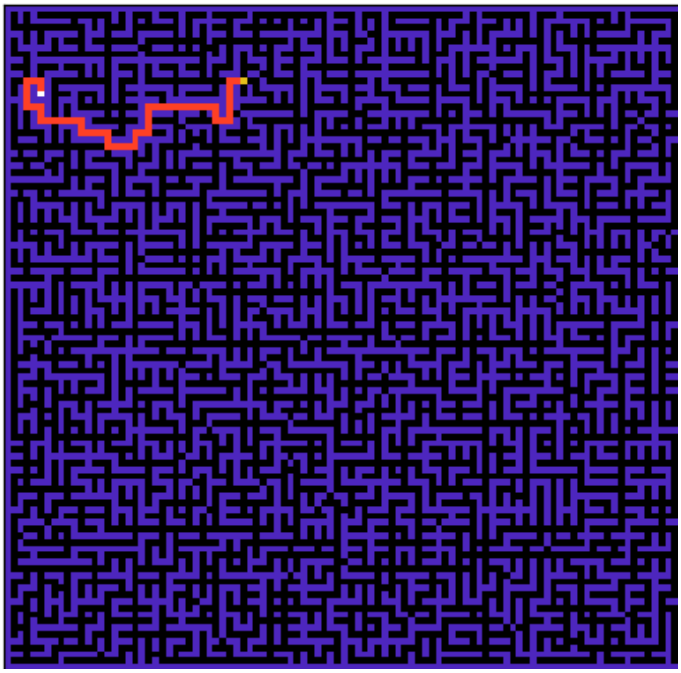

Figure 1: Output of the code (using VI)

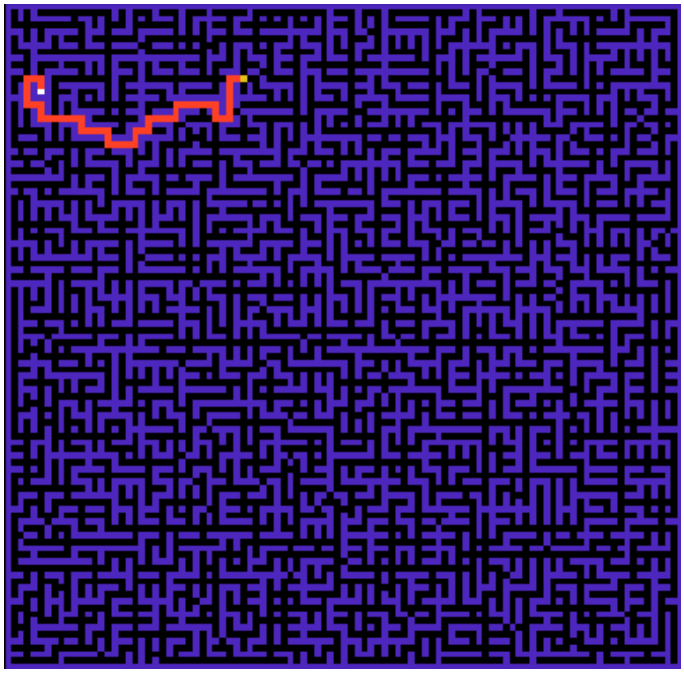

Figure 2: Provided solution

Note that the path from output differs from the solution provided. But the length remains same as the shortest path. (The only difference is in traversing the small rectangle from corner to corner, but in a rectangle, the lengths of both paths joining diagonally opposite points are same).

## Additional Details:

1) Though necessary for certainty, we can omit setting the value function of each end state zero after every iteration, since, the terms in Bellman equation's RHS implicitly make sure they're zero.
2) For the mazes, particularly ones with number of states >700, LP solver is seen to suffer in precision after 2-3 decimals. So, it is assumed that it's an inherent limitation of the solver.
3) For the bigger mazes, particularly grid100, it may take upto 6-7 minutes for the solver to Output the result (though this observation can merely be the result of lack of free RAM).