CS 747 Programming Assignment 2

Parth Shettiwar: 170070021

October 2020

Contents

1 Overall Implementation

There are total 3 functions in planner.py, namely:

```
1) vi
2) hpi
3) lp
```

In all three functions, the final Policy for each state is computed as the minimal index of action which has the maximum Q value (in other words, In case 2 actions give the same Value for a state, choose the smaller index action)

2 Value Iteration

- Implemented by function vi in code.
- The tolerance is set to be 1e-12.
- Vectorization is used to optmize the code
- Algorithm as mentioned in lectures

3 Howards Policy Iteration

- Implemented by function hpi in code.
- Initialisation of Policy by zero i.e. at start, from any state, 1st action will be taken(or action enumerated as 0 in file).
- Using Pulp CBC CMD, the value function is calculated for all states at start by solving the n linear equations, where n is number of states. Subsequently, Q values are computed for actions and for all states.
- For each state, all improvable actions are noted and the first action (which has a lower index as per the enumeration given in data file) whose Q value is greater than the current Q value for that state is chosen for that state.
- Process is repeated until no improvable actions are found, which then sets a flag and loop is exited.

4 Linear Programming

- Implemented by function lp in code.
- Algorithm as mentioned in lecture. Also uses the Pulp CBC CMD, this time to solve the linear inequalities. Minimizes the sum of Value functions for states.
- Total linear inequalities = Number of states * Number of actions
- Achieves the accuracy of Value for each state upto 5 decimals(sometimes there is some error at 6th decimal, this is due to inefficiency of Pulp CBC CMD Solver)

5 Maze Algorithm

The Maze algorithm is described as an episodic problem. Following are the setting for various parameters needed. Also the algorithm works in shortest time for my Value iteration implementation. Hence would want the Maze algorithm to be tested on **vi algorithm**. For rest algorithms, it takes relatively more time.

- The number of states are the total number of grid tiles (both empty and non-empty including start and end states).
- Number of actions is set to 4, as we can only go North, East, West or South from a tile.
- The enumeration of states is done from Left to Right, Top to Bottom, starting from top left corner of grid. So for a grid of 10*10, the top left corner is state 0, top right corner is state 9, bottom left corner is state 90 and bottom right corner is state 99.
- Transition Matrix is hence of size [Number of Grid tiles, 4, Number of Grid tiles]. For a given state, action and new state reached from that action, the value is set to 1, only if the new state is reached from that action. For example, the transition value for the case when the new state is just to the right of previous state(as seen on grid) and the action is East, then the value is set to 1. For rest all transitions from a state to any other state, value is set to 0. In short, corresponding to each state in Transition Matrix and for a particular action, only one value will be set to 1 (which is corresponding to the state reachable from the current state using the action). This is irrespective if the new state is wall or empty. Above all is true only if current state is not wall and not end state. If the current state is a wall or end state, then no transition possible, i.e. all values will be set to 0(becomes a terminal state).
- Reward matrix is initialised with 0. For all transitions into the wall, i.e. where new state is a wall, a high negative reward of -100 is set. Further if new state is an end state, set the reward to be 100. For any other transition from a state, the reward is set to be a constant = 1. The rewards are set to these values only if transition probability is not equal to 0 for a particular tuple of state, action and next state.
- Gamma is set to be 0.9.
- After encoding these variables, the vi function outputs the policy and then we simply start from the start state and output actions according to the policy at each state until we reach any of the end state.

6 Observations

- The grid problem takes huge amount of resources (both time and RAM on grid 80 and higher instances. This is due to enormous number of states as formulated. The fastest algorithm is the value iteration one.
- Even if multiple end states are there, shortest path from start to end state is found using the above formulation.
- Gamma is not set to 1 in task 2, as it entails more complications when multiple end states are there and final policy starts oscillating around 2 neighbouring states. This happens when any particular state is not able to reach the end state (the problem then doesn't become an episodic problem as no path to reach end state, hence probability 0)
- For task1, all computations are done well under 1 minute for any instance.

7 Conclusion

All algorithms have been implemented, specifically Value iteration, Howard Policy Iteration and Linear Programming. Pulp CBC CMD was used for solving the linear equations and inequalities. The Maze problem was solved by formulating it appropriately. Many functions have been optimized, however could have been optimized more to run in a lesser time.