CS533: Intelligent Agents and Decision Making

Assignment 1: Optimizing Finite-Horizon Expected Total Reward

You can use any programming language for the project, including MATLAB. In addition to the write-up materials indicated below, you should also send all of your code to me when you submit the assignment. Submit the project through CANVAS.

1 Part I: Build a Planner

Implement an MDP planning algorithm for optimizing finite-horizon expected cumulative reward. In particular, you should implement finite-horizon value iteration. The input to your algorithm should be a text file that describes an MDP (see below) and a time horizon H (positive integer). The output should be an optimal non-stationary value function and non-stationary policy for the MDP and time horizon.

The input format for the MDP should be a text file with the following format:

- First line gives two integers n and m specifying the number of states and actions respectively.
- After the first line there will be a blank line and then a sequence of m $n \times n$ matrices (each separated by a blank line) that give the transition function for each action. Specifically, the i'th matrix gives the transition function for the i'th action. Entry j, k in that matrix (j is the row and k is the column) gives the probability of a transition from state j to state k given action i. The rows, thus, will sum to 1.
- After the final transition matrix there will be a blank line followed by row of n real numbers. The i'th real number specifies the reward for the i'th state.

An example of the format is below:

```
3 2

0.2 0.8 0.0
0.0 0.2  0.8
1.0 0.0  0.0

0.9  0.05 0.05
0.05 0.9  0.05
0.05 0.05 0.9

-1.0 -1.0 0.0
```

The MDP has 3 states and 2 actions. We see that in state 1 if we take action 1 then there is 0.2 probability of remaining in state 1 and 0.8 probability of a transition to state 2 (zero probability of going to state 3). The reward is negative, except for when the system is in state 3. So the goal should be to get to state 3 and stay there as much as possible. This will generally involve taking

action 1 if not in state 3 and then taking action 2 when in state 3. (see if you can understand why that is the best policy and test that your algorithm can figure that out)

The output format could be two $n \times H$-dimensional matrices, one for the non-stationary value function (column i gives the value function with i steps-to-go) and one for the policy (column i gives the policy for i steps-to-go).

2 Part II: Create Your Own MDP

Here you will test the algorithm on a simple MDP of your choosing. Design a simple MDP where it is easy to see the optimal policy and where the optimal policy depends on the time horizon (i.e. the optimal action at some states will depend on the horizon). The only constraints on the MDP is that that it have at least 20 states and have some probabilistic transitions (i.e. not all actions can be deterministic). The MDP should also have a meaningful description, which will help to interpreting the policy.

Run your planning algorithm on the MDP for at least two different values of the horizon time. Note that for two different horizons H_1 and H_2 such that $H_1 < H_2$, the non-stationary value function for the H_2 case will simply be an extension of the H_1 case.

You should produce a concise write-up that describes the MDP and describes the results of running your algorithm. In particular, you should show the optimal value function and policy, demonstrating that the algorithm is producing the correct results.

3 Part III: More Testing

The instructor will provide you with one or more reasonably sized MDPs in the input format specified above. You should run your code on those MDPs and provide the resulting policies and value functions for a horizon of 10.

IMPORTANT NOTE: Your next assignment will be very similar, but will involve optimizing policies for infinite horizon discounted reward, as well as producing results on an MDP of my choosing. As you will see,the Bellman backup operation can be used in that setting as well. Thus, it will be best for you if you at least write the Bellman backup operation as a generic function, allowing you to reuse it later.