CS533: Intelligent Agents and Decision Making

Assignment 2: Optimizing Infinite-Horizon Discounted Reward with Application to Optimal Parking

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 directly to me via email at afern@eecs.oregonstate.edu. Make sure that you receive an acknowledgement that the instructor received your assignment.

In this project you will implement an MDP planning algorithm and apply it to the problem of "optimal parking".

1 Part I: Build a Planner

Implement an MDP planning algorithm for optimizing expected infinite-horizon discounted cumulative reward. We learned three such algorithms, value iteration, policy iteration, and modified policy iteration. You are free to choose which one to implement. The input to your algorithm should be a description of an MDP and a discount factor. The MDP input format should be the same as used for HW1. The output should be an (approximately) optimal value function and policy for the MDP and discount factor. The output format could be two n-dimensional vectors, one for the value function and one for the policy.

2 Part II: Run the Planner

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 discount factors equal to 0.1 and 0.9. If you are using an algorithm that does not necessarily result in an optimal policy, you should provide a bound on how suboptimal the resulting policies are. You can refer to the course notes for this purpose—recall, that this bound can be computed in terms of the "Bellman error" across iterations.

3 Part III: Parking Domain

Most people like to park close to wherever they are going. This desire often seems to lead to irrational behavior—for example, driving around a parking lot for several minutes looking for a slightly closer spot. Here we will see what an MDP planner does in this scenario.

You should first design a simple MDP to represent the experience of parking a car. The MDP should capture the following qualitative characteristics of the parking problem:

1. Parking spots closer to the store are more desirable to the agent.
2. The probability that a parking spot is available is smaller the closer a spot is to the store.
3. It is undesirable to spend much time searching for a spot (i.e. driving around the parking lot).

4. The parking lot should have two rows of parking spaces A and B (see below figure) that must be traveled in a loop. These rows are parallel to each other and have n parking spots labeled $A[1], \ldots, A[n]$ and $B[1], \ldots, B[n]$. $A[1]$ and $B[1]$ are closest to the store, $A[n]$ and $B[n]$ are furthest from the store. In row A the agent can only drive toward the store (i.e. move from $A[i]$ to $A[i-1]$) unless the agent is in $A[1]$ where it can only move to $B[1]$. When the agent is in row B it can only move away from the store (i.e. move from $B[i]$ to $B[i+1]$), unless the agent is in row $B[n]$ where it can only move to $A[n]$. Thus the agent can only drive in a "circular" motion around rows A and B.

```
  STORE

A[1]  B[1]
A[2]  B[2]
 ........
A[n]  B[n]
```

5. The parking spots closest to the store $A[1]$ and $B[1]$ are handicap spots. There is a high cost (negative reward) for parking in these spots (although they are desirable with respect their closeness to the store). Also the probability that the handicap spot is available should be high.

6. If an agent attempts to park in a spot that contains a car, then there is a high cost (negative reward) since there will be a collision.

7. A parking trial ends when the agent decides to park, resulting in either a collision or a parked car. This means that the MDP will transition to a terminal state that it stays in forever (all actions result in no transition).

You need to specify an MDP that roughly represents the above features. Here is a suggested structure, but you are free to try something else.

- **State Space:** Each state can be viewed as a triple (L, O, P) where L is a location (one of the $A[i]$ or $B[i]$), O is a boolean variable that is TRUE if the spot at that location is occupied by another car, and P is a boolean variable that is TRUE if the agent is parked (or tried to park) at the location. Thus there will be $8n$ states since there are $2n$ locations and two values of A and B. Initially P will be FALSE and the trial ends when P is true. That is, any state where P is true is considered to be a terminal state.

- **Actions:** There are three actions PARK, DRIVE, and EXIT. When the action DRIVE is taken the agent is moved to the next parking spot (according to the circular driving pattern described above) and a coin is flipped to set the value of O. The probability that O is true should increase for spots closer to the store (the details of this are up to you). However the probability that O is true for handicap spots $A[1]$ or $B[1]$ should be very low. The DRIVE action does not change the P variables (P is initialized to be FALSE). When the action PARK is taken the value of P is set to TRUE and L and O are left unchanged. The action exit does not change the state for any state where P is false. When P is true the action EXIT causes the trial to terminate and will always be the last action taken.

- **Reward:** States where P is FALSE will get a negative reward that represents the cost of driving. This way if the agent drives for a long time it will accumulate negative reward. Thus,long driving times will (eventually) look undesirable.

  For terminal states (any state where P is TRUE) the reward should be based on the location and the value of O. Clearly if O is TRUE then we want there to be a large cost, since this corresponds to a collision. If O is FALSE then we want the reward to based on two factors. There should be more reward for parking closer to the store, but parking in $A[1]$ or $B[1]$ (the two closest spots) should be discouraged by giving a smaller reward (since they are handicap spots).

- **Discounting:** It is most natural to use a discount factor close to 1 for this problem so that the behavior is mainly influenced by the MDP dynamics and reward.

You should produce code that can take certain parameters that characterize a parking problem (e.g. the various rewards and probabilities) and produces a corresponding MDP in the input format of your planner. You will also want to be able to interpret the policy and value function output by the planner in the context of the parking problem, so you may need to convert the output of the planner to a meaningful representation for you.

Select a value of n (say around 10) and select two sets of parameter values. This will result in completely specifying two MPDs (one for each set of parameter values). Select the parameters so that you believe that the optimal behavior in each MDP is different. Run your planner on the two MDPs.

You should produce a brief write-up that describes your MDP design and the two specific MDPs given to your planner. Describe the optimal policy found by the planner and whether it agreed with your intuition or not. Also describe the value function and indicate which state(s) had the highest value.