CS533: Intelligent Agents and Decision Making
Mini Project 2: Reinforcement Learning for Rational Parking
Winter 2012

You may work with one partner on this project if you choose to.

Most people like to park close to wherever they are going. This desire often seems to lead to irrational behavior—e.g. driving around a parking lot for several minutes looking for a slightly closer spot. In this project you will put a reinforcement learning agent in this situation and observe its behavior.

Part I: Designing the MDPs

NOTE: The MDP description is just like the one for mini-project 1, just repeated here for your convenience. The key distinction for this project is that Part I asks you to design a simulator for the environment.

The first part of the project is to design simple MDPs to represent the experience of parking a car. The MDPs will be very similar, differing primarily in the exact parameter values you choose. You will then design a simple simulator for these environments that your reinforcement learning agent can act in. 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. When a parking trial begins the agent is randomly placed at either $B[1]$ or $A[n]$. A trial ends when the agent decides to park, resulting in either a collision or a parked car.

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 is 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 high. 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:** You can use a discount factor of 1 (no discounting). However, it will be important to put a horizon limit on the trials, otherwise you might end up driving around the parking lot for a very long time during learning.

From this discussion the total reward over a trial is the sum of the number of driving steps plus the parking reward. An agent needs to learn how to balance the time spent searching for a better spot with the location of the spot. The rational balance will depend on the relative cost of driving versus parking further away.

Your job is to create a simulator for the "Parking MDP". Ideally this would allow you to easily change the value of n and the various parameters of the MDP (rewards and probabilities). There is no need to provide a nice user interface, it is fine to recompile when you make such changes. The simulator should allow you to easily interface a learning or non-learning agent to it and run many trials to collect the average reward.

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.

The simulator need not be fancy. The basic program is quite simple and should not be a huge programming effort.

PART II: Policy Simulation

Here you will measure the performance of some basic policies, which will be used as baselines. You will collect these measurements for both of the MDPs you specified in step 1. To measure the performance of a policy you can simply run many trials of the policy (e.g. 1000) and then average the total reward of the trials.

1. You should measure the performance of a random policy on this MDP. This policy should select the PARK action with some probability p and DRIVE with probability $1 - p$. You can specify p.

2. You should measure the performance of the policy that DRIVES whenever O is TRUE and when O is FALSE selects PARK with probability p and otherwise DRIVES with probability $1 - p$.

3. Try to write your own SIMPLE policies (one for each MDP) that improves on the ones above and evaluate it. You do not need to work out "optimal policies", just implement one or two obvious improvements of the above.

PART III: Reinforcement Learning

Implement a reinforcement learning agent for this environment. (You can choose model-based or model-free such as Q-learning.) Your goal is to measure the performance of the agent as it learns. I would suggest allowing the agent to learn for N trials and then measure the performance of the resulting greedy policy (as you did in PART II for the random policies), then run another N trials of learning and measure the performance, continue this process of learning and evaluation until the performance does not improve. You will want to select N so that you can see some change in performance.

You will need to select an exploration policy for the agent. One such policy could be ϵ-greedy exploration where you take a random action with probability ϵ and otherwise take a greedy action (with probability $1 - \epsilon$). A typical value might be $\epsilon = 0.1$, but you may want to play with the exact value. However, you are free to use whatever exploration strategy you want, provided that it leads the agent to improve its performance. (In my experience, students have generally found that the learning algorithm is able to beat their own hand-built policies.)

You should evaluate the learning performance for different parameters of your algorithm. For example, if your algorithm involves a learning rate (α), then consider using a "small" constant value, a "larger" constant value, and an dynamic α equal to $1/n$ (where n is the number of updates).

Note that you will want to limit the number of steps for each trajectory to avoid an infinite trial.

What You Need to Produce

1. Provide me with the code for you implementation of the simulator and learning agent.

2. Provide a BRIEF writeup that gives:

   (a) Your choices in Part I

   (b) The performance of the policies in part II for each MDP (describe the policies that you designed)

   (c) The performance of the learner for the different learning rates. Be sure to specify what exploration policy you used. Compare the learners performance to the policies from part II. Describe the learned behavior of the agent for each MDP. Were the behaviors different? Where they as expected?

Email this material directly to me at afern@eecs.oregonstate.edu