6.S078 Planning Algorithms, Fall 2014

November 19, 2014

Assignment 6 (due Wed. Dec 10 before class)

In this assignment, you will implement some of the major algorithms for computing optimal policies in Markov decision processes (MDP's) and experiment with the performance of the algorithms. You will also implement one simple algorithm for POMDPs.

Implementation

I have uploaded a zip file that includes one Python file `mdp.py` and several data files, with `mdp` extensions. Note that you will need to install `cvxopt`, from `http://cvxopt.org/` (or some alternative) to solve linear programs. Look at the documentation on dense matrices and Cone Programming/Linear Programming.

The file `mdp.py` defines a simple `MDP` class and provides a function for reading in definitions from files. There are also skeletons for the code that you need to write, namely implementations of value iteration, (modified) policy iteration and linear programming solvers for MDPs. Chapter 17 of AIMA has good pseudo-code for the value iteration and modified policy iteration. The LP formulation is in the lecture slides. None of the programs are very long. I understand that you can probably find implementations on the Web, but it's really worth implementing them yourself to make sure that you understand these algorithms.

There are two simple MDPs provided to help you debug:

- `test.mdp` - described below. The results for this mdp are in the code file.
- `fig17.mdp` - the simple grid world example in Chapter 17 of AIMA, with a movement noise of 0.1. The results for this mdp are in the code file.

Hand in your code in runnable form.

Performance

I have given you three large mdp files (in compressed form) to explore performance issues. The files encode the "cat and mouse" game described below. Perform the following experiments on at least one of these large MDPs (indicate which one(s) you used); remember to uncompress the files before using.

1. Explore the impact of the number of iterations of policy evaluation on the accuracy of the policy (compare to the policy from value iteration).

2. Compare the running time of value iteration, modified policy iterating (for various values of number of iterations) and linear programming.

3. Do these experiments for at least two values of gamma: 0.9 and 0.95.

Show your results and summarize them in your write up.

POMDP

Implement Monahan's algorithm for finite-horizon POMDPs. This is composed of an enumeration phase, where the alpha vectors from time step $i+1$ are generated from those at time steo i, followed by a pruning phase (implemented via linear programming) that prunes alpha vectors that cannot be optimal for any belief state.

- Illustrate your algorithm on the classic Tiger problem described in class. You can simply encode the problem in code directly.
- Plot the pruned alpha vectors for the first four time steps. Describe how the alpha vectors partition the belief space.
- Provide statistics on the number of alpha vectors generated during the enumeration phase and how many are left after pruning.

Please read section 4.4 of Tony Cassandra's thesis (link is on Stellar) describing Monahan's algorithm, particularly sections 4.4.2 and 4.4.3 on the "Reduction Phase" (pruning).

MDP Reference

MDP file format

Each line of an MDP description file has one of three possible formats. The first possibility is that the line has the simple form

```
state
```

where `state` is any string (no white space). Such a line indicates that `state` is the start state of the MDP. The second possibility is that the line has the form

```
state reward
```

where state is as before and reward is a number. Such a line indicates that the reward in `state` is `reward` If the state is terminal, this is indicated by a line such as

```
state reward Terminal
```

The last possibility is that the line has the form

```
fromState action toState prob
```

where the first three tokens are strings, and `prob` is a nonnegative number. Such a line indicates that the probability of transitioning from `fromState` to `toState` under action `action` is `prob`. Multiple lines of this form with the same `fromState` and `action` can be combined. Thus,

```
u a v 0.3 u 0.4 w 0.1
```

is equivalent to the three lines

```
u a v 0.3
u a u 0.4
u a w 0.1
```

Lines of these forms can appear in the file in any order. However, if a reward is given more than once for the same state, then the *last* reward value given is assigned to that state. On the other hand, if more than one probability is given for the same state-action-state triple, then the *sum* of these probabilities is assigned to this transition. If the probabilities of transitioning out of a given state under a given action do not add up to one, then these numbers are renormalized so that they do. However, if this sum is zero, an exception is thrown. States that are not given a reward in the description file are assumed to have a reward of zero; likewise, transitions that do not appear in the file are assumed to have zero probability. An exception is also thrown if no start state is declared (using the first form above); if more than one such line appears, then the last one is taken to define the start state.

For instance, the following encodes an MDP with five states, `-2`, `-1`, `0`, `+1` and `+2`. There are two actions, `L`(eft) and `R`(ight) which cause the MDP to move to the next state to the left or right 90% of the time, or to move in the opposite direction 10% of the time (with movement impeded at the end points `-2` and `+2`). The start state is `0`. The reward is 1 in `0`, -1 in `-2`, and -2 in `+2`. The start state is `0`.

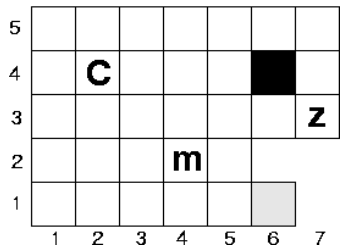
```
0
0 1
0 L -1 0.9  +1 0.1
0 R +1 0.9  -1 0.1
-1 L -2 0.9 0 0.1
-1 R  0 0.9 -2 0.1
-2 -1
-2 L -2 0.9 -1 0.1
-2 R -1 0.9 -2 0.1
+1 R +2 0.9 0 0.1
+1 L  0 0.9 +2 0.1
+2 -2
+2 R +2 0.9 +1 0.1
+2 L +1 0.9 +2 0.1
```

Theis mdp is in `test.mdp`. The optimal actions and values for each state are given in the file.

In this case, as expected, the optimal policy says to move to the center (right if in `-1` or `-2`, and left if in `+1` or `+2`), and to move left (away from the larger penalty state `+2`) if in the center.

Playing cat and mouse

A mouse who lives with a cat in the following grid world:



The mouse (**m**) can occupy any of the 31 blank squares. The cat (**C**) also can occupy any square, except for square (6,1) which is the mouse's hole (too small for the cat to squeeze in). There is cheese (**z**) that, at any given time, can be found in one of the following four squares: (2,3), (4,1), (5,5) and (7,3). Thus, this MDP has 31*30*4=3720 states.

The cat and the mouse can each move one square in any direction – vertically, horizontally or diagonally. They also can choose not to move at all. Thus, there are nine possible moves from each square. If an action is attempted that causes the creature to bump into a wall, then it simply stays where it is.

In this problem, we always take the point of view of the mouse. When the mouse is on the cheese, it receives a reward of 1. When the cat is on the mouse, it (the mouse) receives a reward of -11. When the cat is on the mouse, and the mouse is on the cheese, the reward is -10. All other configurations have a reward of 0. Thus, the mouse is trying to eat cheese while simultaneously avoiding the cat.

We will consider three different cats. The first cat, poor thing, is blind and oblivious, and simply wanders randomly around its environment choosing randomly among its nine available actions at every step. The second cat is hungry, alert and unrelenting. This cat always heads straight for the mouse following the shortest path possible. Thus, after the mouse makes its move, the cat chooses the action that will move it as close as possible to the mouse's new position. (If there is a tie among the cat's best available options, the cat chooses randomly among these equally good best actions.) However, when the mouse is in its hole and out of sight, the cat reverts to aimless (i.e., random) wandering. The third cat is also alert, but has a more sporting disposition, and therefore follows a combination of these two strategies: half the time, it wanders aimlessly, and half the time, it heads straight for the mouse. Machine-readable (and compressed) descriptions of the three MDP's corresponding to these three cats are provided in the files `oblivious.mdp.zip`, `unrelenting.mdp.zip` and `sporting.mdp.zip`.

There is always cheese available in exactly one of the four locations listed above. At every time step, the cheese remains where it is with 90% probability. With 10% probability, the cheese vanishes and reappears at one of the four locations, chosen randomly.

States are encoded as six tuples, the first two numbers indicating the position of the mouse, the second two numbers the position of the cat, and the last two numbers the position of the cheese. Thus, 4:2:2:4:7:3 indicates, as depicted in the figure above, that the mouse is in (4,2), the cat is in (2,4), and the cheese is in (7,3). The cat and the mouse alternate moves. However, in encoding the MDP, we collapse both moves into a single state transition. In addition, the cheese, when it moves, does so simultaneously with the mouse's move. For instance, from the configuration above, if the mouse moves to (5,2) and the cat responds by moving to (3,3), while the cheese moves to (5,5), this all would be encoded as a *single* transition from state 4:2:2:4:7:3 to 5:2:3:3:5:5. Actions in the MDP refer to action's that the mouse can make; the cat's actions are effectively "hard-wired"; into the dynamics of the MDP itself.

Acknowledgments

Thanks to Rob Schapire (Princeton) for the description of the cat and mouse problem and the mdp definitions. Schapire thanks Dale Schuurmans for the main idea for the assignment.