ROB311: Artificial Intelligence

Project #4: Markov Decision Processes

Winter 2020

Overview

In this project, you will further explore learning algorithms by working with two closely-related techniques for sequential decision making: *value iteration* and *policy iteration* for Markov decision problems. The goals are to:

- understand value iteration and the use of the Bellman update equations; and
- compare value iteration with policy iteration for the same problem domains.

The project has two parts, worth a total of **50 points**. All submissions will be via Autolab; you may submit as many times as you wish until the deadline. To complete the project, you will need to review some material that goes beyond that discussed in the lectures—more details are provided below. The due date for project submission is **Tuesday, April 7, 2020, by 11:59 p.m. EDT**.

As in Project #3, each part already has some basic code in place for you to start with. There are three types of files provided to you in the handout package: support files (named `mdp_*.py`), template files (named `part*.py`) and test files (named `test*.py`). The students are only required to complete and submit the template files. The test files can be used to run preliminary tests on your function implementations.

Part 1: Markov Decision Processes via Value Iteration

Value iteration is a well known method for solving Markov decision problems (processes). This iterative technique relies on what is known as the 'Bellman update' (see original work by Bellman in 1957), which you will code up as part of the project. Your tasks are to:

1. Write a short function, `get_transition_model()`, that generates the state transition model (matrix) for the simple cleaning robot problem described by Figure 1. This transition model will be needed to solve an instance of the cleaning robot MDP (see next bullet).
2. Implement the value iteration algorithm given in AIMA on pg. 653, which accepts an MDP description as input (states, possible actions, transition matrix, rewards, discount factor) and produces an epsilon-optimal policy (and a utility function). The function, `value_iteration()`, will also accept a threshold (ϵ) and a maximum number of iterations to run.

You will submit the completed template files `part1_1.py` and `part1_2.py` through Autolab. We will test your code on several problems, including on the grid world example given in AIMA on pg. 646! We have included an environment file (`mdp_grid_env.py`) in the handout package that defines the variables which are required for the AIMA problem. The function `init_stochstic_model()` (i.e., the function that generates the transition model) has deliberately not been implemented, so as not to give away all the test cases. After implementing this function, you should be able to test your solution on more complex problem instances, as done in Autolab. Thus, you have the option to create additional tests for the grid environment using the provided test files—this should help with debugging, etc.

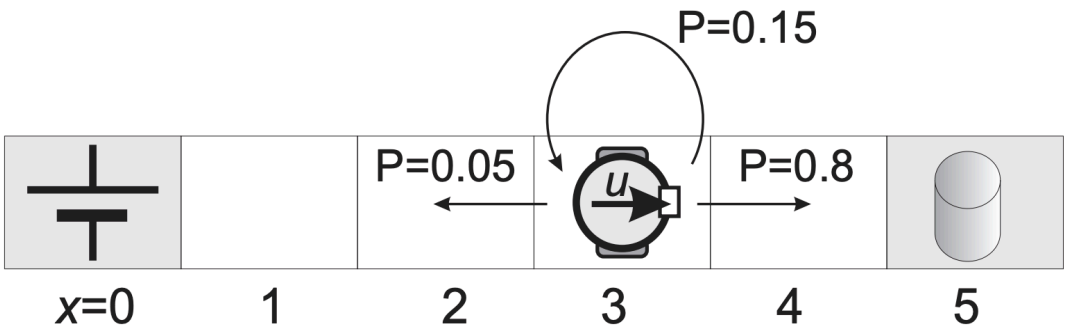

Figure 1: A simple cleaning robot problem. There are six states; the robot wants to put rubbish in the bin (State 5) and to recharge its batteries (State 0). Both of these states are terminal states (i.e., once reached, the robot does not leave the state). The robot may choose to move left or right to an adjacent cell. Due to environment uncertainties, such as a slippery floor, for example, state transitions are not deterministic: when trying to move in a certain direction, the robot succeeds with a probability of 0.8; with a probability of 0.15 it remains in the same state, and with a probability of 0.05 it may move in the opposite direction. The reward in State 0 is 1, in State 5 is 5, and is zero otherwise.

Part 2: Markov Decision Processes via Policy Iteration

As discussed in the lectures, value iteration is not the only way to solve an MDP; another popular alternative is policy iteration. The policy iteration framework is different than that of value iteration: we begin with an initial, sub-optimal policy (possibly random), and then refine that policy. Your task is to:

1. Implement the policy iteration algorithm given in AIMA on pg. 657, which accepts an MDP description as input (states, possible actions, transition matrix, rewards, discount factor) and produces an optimal policy (and a utility function). The function, `policy_iteration()`, will also accept a variable that specifies the maximum number of iterations to run.

You will submit the completed template file `part2.py` through Autolab. We will test your code on several problems, including on the grid world example given in AIMA on pg. 646, as above.

Grading

We would like to reiterate that **only** the functions implemented in the template files will affect your marks. Points for each portion of the project will be assigned as follows:

- Value Iteration – **30 points** (1 test: 5 points; 3 tests: 5 points, 10 points, and 10 points)

  The first test (Part 1A) will evaluate your state transition model, to ensure that you understand how to write down such models from a high-level description (i.e., that given in the caption to Figure 1). The next three tests (Part 1B) will evaluate your value-based MDP solver, first for the simple 1-D cleaning robot world, and then for the grid world defined in AIMA (with some tweaks).

- Policy Iteration – **20 points** (2 tests $\times$ 10 points per test)

  The two tests will evaluate your policy-based MDP solver on different types of grid worlds.

Total: **50 points**

Grading criteria include: correctness and succinctness of the implementation of support functions, proper overall program operation, and code commenting. Please note that we will test your code *and it must run successfully*. Code that is not properly commented or that looks like 'spaghetti' may result in an overall deduction of up to 10%.