MIE567H1: Dynamic Distributed Decision Making

Project #1: Markov Decision Processes and Dynamic Programming

Deadline: Monday February 11, 2020, 11:59 pm on Quercus
Maximum Points: 100

Instructions: Please read all instructions carefully before attempting each part of the problem. All programming should be done in Python - please ensure that your code runs on a recent version of Python (at least 3.5). You need to implement all the algorithms yourself, and you are allowed to use only standard packages.

In this project, you will implement and compare the value and policy iteration algorithms in order to solve a variation of the Gridworld domain. Gridworld is a standard benchmark problem that serves as a good model for teaching robots to navigate in their environment.

You should provide a **ZIP folder** containing your report in **PDF format** and your Python code. The report should include detailed answers to all questions, including all derivations, analysis and relevant figures. You should include all the code necessary to run your experiments.

Problem Description: Figure 1 (below) shows a rectangular grid world representation of the problem. The cells of the grid correspond to the states of the environment. At each cell, four possible actions exist, each corresponding to one compass direction (e.g. north, east, west, south), which deterministically cause the agent to move one cell in the respective direction on the grid. Actions that would take the agent off the grid leave its location unchanged, but also result in a reward of -1. Other actions have zero reward, except those that move the agent out of the special states A and B. From state A, all four actions yield a reward of +10 and take the agent to A'. From state B, all actions yield a reward of +5 and take the agent to B'.

Modelling [20 points]: First, you are tasked with modelling the Gridworld domain above as a Markov decision process. Then, you are asked to provide a complete programming description of the problem that will be used to solve it computationally.

1. [10 points] Explain how you would model this navigation problem as a Markov decision process. In particular:
   (a) [2 points] Why is this problem an MDP?
   (b) [2 points] What are suitable state and action spaces for this problem? Are these the only possible choices? Why or why not?

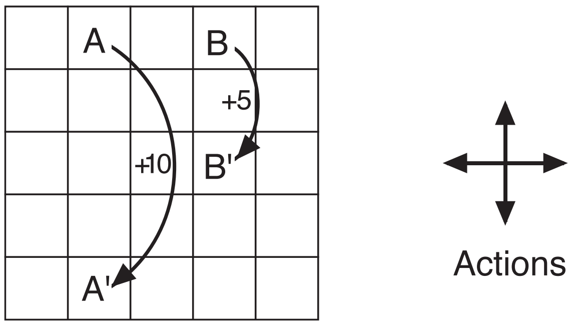

Figure 1: Gridworld domain

(c) [2 points] What is the transition probability matrix P? (You may describe just the non-zero entries.)

(d) [2 points] Is the reward function provided the only possible one? If so, explain why. If not, provide an example of a different reward function that would lead to the same optimal behaviour.

(e) [2 points] Derive the discounted Bellman equation for the problem, and simplify as much as you can. (Hint: to avoid deriving a separate value for each state, try to find groups of states such that you can write a single expression for V for them) What do you think is/are the optimal policy/policies for this problem, and why (you do NOT need to solve the Bellman equations)?

2. [10 points] Now, in a Python file called `Gridworld.py`, create a class that replicates the behaviour of the MDP you formulated in the previous question. Your class should contain four functions: one to return the initial state of the MDP, one to return a view of all possible states, and two to return, respectively, the reward and probability of a transition (s, a, s') from state s to state s' when taking action a.

Policy Evaluation [20 points]: Now, suppose the agent selects all four actions with equal probability. Use your answers to questions 1 and 2 to write a python function, in a new file `policy_evaluation.py` to find the value function for this policy. You may use either an iterative method or solve the system of equations. Show the value function you obtained to at least four decimals.

Value Iteration [20 points]: The goal now is to solve the MDP above using dynamic programming. In a separate file called `value_iteration.py`, provide a complete implementation of the value iteration algorithm you learned in class for solving the Bellman equations you derived earlier. Ideally, you should stop updates when the maximum change in the values between iterations drops below 10^-6. Include all the relevant plots of convergence in your report. You should also run your algorithm for different discount factor $\gamma \in [0.8, 0.95, 0.99]$ and compare the plots. Provide a complete analysis of your results. In particular, answer the following:

- Did your algorithm converge at all?
- How many iterations did this take for each value of γ?
- What is the best value of γ? Also, What was the final policy you obtained for each value of γ?
- Does the optimal policy you obtained correspond to the policy you conjectured earlier?

Policy Iteration [20 points]: Repeat the previous part of the assignment, but now implement the policy iteration algorithm from class in a file called `policy_iteration.py`. Report your results and answer all questions as in the previous part.

Comparison of Algorithms [20 points]: In the final section of your report, you must compare the two algorithms you implemented and their performance on the Gridworld domain, and comment on any differences you observed. In particular, please answer at least the following questions in your report:

- Compare the performance (e.g. values) of the optimal policies obtained using value and policy iteration to the performance of the policy that chooses an action at random (as you analyzed earlier), and comment on the difference.
- Were the value functions and policies you obtained, and the number of iterations required to obtain these policies, similar between algorithms? How do your results differ for different values of the parameter(s) (e.g. γ)?
- Which algorithm was more difficult to implement and why? Which algorithm do you think would work better if the problem was scaled up?
- Include any additional insights, challenges, or important observations that you discovered while building or running your experiments.