CS 747: Programming Assignment 2

(TA in charge: Pragy Agarwal)

In this assignment, you will implement algorithms for finding an optimal policy for a given MDP. The first part of the assignment is to apply Linear Programming (LP), based on the formulation presented in class. The second part of the assignment is to implement three different variants of Policy Iteration (PI) that were also presented in class: namely Howard's PI, Mansour and Singh's Randomised PI, and Batch-switching PI. Thereafter, you will compare the efficiency of these variants of PI by running a set of experiments.

Data

This directory provides a few samples of input and output that you can use to test your code. The directory contains three MDPs encoded as text files, with each file in the following format.

```
Number of states
Number of actions
Reward function
Transition function
Discount factor
```

In these files, and also in the MDPs on which your algorithms will be tested, the number of states S will be an integer greater than 0 and less than 100. Assume that the states are numbered 0, 1, 2, …, (S - 1). In this assignment, you will only be tested on MDPs that have *exactly* two actions, which are numbered 0 and 1. Thus, the number of actions A is 2. The reward function will be provided over S×A lines, each line containing S entries. Each entry corresponds to R(s, a, s'), wherein state s, action a, and state s' are being iterated in sequence from 0 to (S - 1), 0 to (A - 1), and 0 to (S - 1), respectively. A similar scheme is adopted for the transition function T. Each reward lies between -1 and 1 (both included). The discount factor is a real number between 0 (included) and 1 (excluded).

Below is a snippet of python code that is used to generate MDP files.

```
print S
print A

for s in range(0, S):
    for a in range(0, A):
        for sPrime in range(0, S):
            print str(R[s][a][sPrime]) + "\t",

        print "\n",

for s in range(0, S):
    for a in range(0, A):
        for sPrime in range(0, S):
            print str(T[s][a][sPrime]) + "\t",
```

```
        print "\n",

print gamma
```

Solution

Given an MDP, your program must compute the optimal value function V* and an optimal policy π^* by applying the algorithm that is specified through the command line. Create a shell script called `planner.sh` to invoke your program. The arguments to `planner.sh` will be

- `--mdp` followed by a full path to input the MDP file,
- `--algorithm` followed by one of `lp`, `hpi`, `rpi`, and `bspi`,
- `--batchsize` followed by an integer in {1, 2, …, S}, and
- `--randomseed` followed by an integer.

Make sure you don't make any assumptions about the location of the MDP file relative to the current working directory; read it in from the full path that will be provided. The algorithms specified above correspond to LP, Howard's PI, Mansour and Singh's Randomised PI, and Batch-switching PI, respectively. The batch size is only relevant for Batch-switching PI, and the random seed only relevant for Randomised PI.

Here are a few examples of how your planner might be invoked (it will always be invoked from its own directory).

- `./planner.sh --mdp /home/user/mdpfiles/mdp-5.txt --algorithm hpi`
- `./planner.sh --mdp /home/user/temp/data/mdp-7.txt --algorithm bspi --batchsize 8`
- `./planner.sh --mdp /home/user/temp/data/mdp-7.txt --algorithm rpi --randomseed 89`
- `./planner.sh --mdp /home/user/temp/data/mdp-7.txt --algorithm lp --randomseed 89`

In the last example, the random seed is irrelevant, and so your program can simply ignore it.

You are free to implement the planner in any programming language of your choice. You are not expected to code up a solver for LP; rather, you can use available solvers as blackboxes (more below). Your effort will be in providing the LP solver the appropriate input based on the MDP, and interpreting its output appropriately. For the three PI algorithms, you are expected to write your own code; you may not use any libraries that might be available for the purpose. For all variants of PI, your initial policy must be one that takes the 0 action from every state.

Output Format

The output of your planner must be in the following format, and **written to standard output**.

```
V*(0)   π*(0)
V*(1)   π*(1)
.
.
.
V*(S - 1)   π*(S - 1)
```

In the `data` directory provided, you will find three output files corresponding to the MDP files, which have solutions in the format above.

Since your output will be checked automatically, make sure you have nothing printed to stdout other than the S lines as above in sequence. If the testing code is unable to parse your output, you will not receive any marks.

Note:

1. Your output has to be written to the standard output, not to any file.
2. For values, print at least 6 places after the decimal point. Print more if you'd like, but 6 (`xxx.123456`) will suffice.
3. If your code produces output that resembles the solution files: that is, S lines of the form

   ```
   value + "\t" + action + "\n"
   ```

   or even

   ```
   value + " " + action + "\n"
   ```

   you should be okay. Make sure you don't print anything else.

4. If there are multiple optimal policies, feel free to print any one of them.

Submission

You will submit two items: (1) working code for your planner, which implements different algorithms, and (2) a report comparing the performance of the different variants of PI.

1. Create a directory titled `[rollnumber]`. Place all your source and executable files in this directory. The directory must contain a script titled `planner.sh`, which must take in the command line arguments specified above, and produce the output also as specified.
2. Generate a large number of MDP instances (say 100) with 50 states, and record the average number of iterations taken by Howard's PI, Randomised PI, and BSPI to terminate (all starting from the "all 0s" policy). Run BSPI with at least 10 batch sizes to get a good sense of the effect of the batch size. Collate this data into a graph or a table. In a file called `report.pdf`, describe the process you used to generate the MDPs, the results obtained by the PI variants (as a graph or a table), and your interpretation of the results (why are they as observed? what is interesting/unexpected in the

results?). Place `report.pdf`, as well as the scripts and data related to your experiments, in the [rollnumber] directory.

Before you submit, make sure you can successfully run `planner.sh` on the departmental (`sl2`) machines. Provide references to any libraries and code snippets you have utilised (mention them in `report.pdf`). It is okay to use libraries for data structures and for operations such as sorting. You may also use libraries for solving systems of linear equations. However, the logic used for policy improvement under PI, and for translating the given MDP into a linear program, must entirely be code that you have written.

Compress and submit your directory as `[rollnumber].tar.gz`. The directory must contain all the sources, executables, and experimental data, and importantly, your `planner.sh` script and `report.pdf` file. Make sure you upload your submission to Moodle by the submission deadline.

Evaluation

Your planner will be tested on a large number of MDPs. Your task is to ensure that it prints out the correct solution (V^* and π^*) in each case, using each of the algorithms you have been asked to implement. 7 marks are allotted for the correctness of your four algorithms, and 3 marks for your report. You will not get the full 3 marks for the report if any of your PI variants has not been implemented correctly.

The TAs and instructor may look at your source code to corroborate the results obtained by your agent, and may also call you to a face-to-face session to explain your code.

Deadline and Rules

Your submission is due by 11.55 p.m., Tuesday, September 5. You are advised to finish working on your submission well in advance, keeping enough time to test it on the `sl2` machines and upload to Moodle. Your submission will not be evaluated (and will be given a score of zero) if it is not received by the deadline.

You must work alone on this assignment. Do not share any code (whether yours or code you have found on the Internet) with your classmates. Do not discuss details of your implementation with anybody else.

You will not be allowed to alter your code in any way after the submission deadline. Before submission, make sure that it runs for a variety of experimental conditions on the `sl2` machines. If your code requires any special libraries to run, it is *your* responsibility to get those libraries working on the `sl2` machines (go through the CSE bug tracking system to make a request to the system administrators).

References for Linear Programming

Although you are free to use any library of your choice for LP, we recommend that you use the Python library `PuLP` (https://pythonhosted.org/PuLP/) or the `lp_solve` program (http://lpsolve.sourceforge.net/5.5/). Both of these are already installed on the `sl2` machines.

`PuLP` is convenient to use directly from Python code: here is a short tutorial and here is a reference.

`lp_solve` can be used both through an API and through the command line. Here is a reference and here is an introductory example.