ECE 4984 & 5984: (Advanced) Robot Motion Planning (Spring 2017)

Homework 4

Due: Tuesday April 11th, 9PM

March 30, 2017

Instructions. This homework constitutes 7% of the course grade. You must work individually on all problems. It is okay to discuss the problems with other students in class. However, your submission must be your original work. This implies:

- If you discuss the problems with other students, you should write down their names in the pdf report.
- If you refer to any other source (textbook, websites, etc.), please cite them in the report at the relevant places.
- The answers in the pdf report must be written entirely by you from scratch. No verbatim copy-paste allowed without citations.
- Any software you submit must be written entirely by you and your partner with no copy-pasting of significant portions of code from other sources.

Please follow the submission instructions posted on canvas exactly. You must submit your assignment online on canvas by the due date. Your submission must include one pdf file with the answers to all the problems and one or more files containing your code. It is okay to scan your answers and create the pdf submission.

Problem 1 — 100 points

Implement (in C/C++/Python/MATLAB) the value iteration algorithm (as discussed in class) to solve a stochastic shortest path problem. The input to your algorithm is a graph given in the `input.txt` file. The format of this file is the same as that for the input file in Homework 1.

- The first line of the input file gives the total number of vertices, n, in the graph.
- The second line gives the starting vertex index (indices go from 1 to n).
- The third line gives the goal vertex index (indices go from 1 to n).
- Starting from the fourth line, we have the edges in the graph specified in the form of an edge list. Each line specifies one edge: `i j wij` which indicates that there is a (directed) edge from `i` to `j` with a cost of `wij`. Note that this is a directed graph.

The set of states, S, for the MDP are the set of vertices in the graph. The set of actions, A, are the set of directed edges. For each state, the available actions are given by the *outgoing* edges from the corresponding vertex. **In addition, you should also add a dummy action for the goal state that makes the robot stay at the goal with probability one and with reward zero.**

Suppose a i has n outgoing edges. Let j be one of the outgoing neighbors. Therefore, ij is one of the available actions for the robot at state i. The transition probability is then given as $T(i, ij, j) = 0.7$ and $T(i, ij, j') = \frac{0.3}{n-1}$ where j' is one of other $n-1$ outgoing neighbor of i. That is, the robot reaches its intended neighbor with 70% probability and reaches each of the remaining $n-1$ neighbors with $\frac{30}{n-1}$% probability.

The reward for a transition from state i to j is equal to the $-wij$.[1]

The output of your program must be a text file `output.txt` that contains the following information:

- The first line in the file specifies the final value function produced by your algorithm. List each $V(s)$ starting with $s = 1$ to $s = n$ separated by a comma. That is, $V(1), V(2), \ldots, V(n)$.
- The second line in the file specifies the optimal policy produced by your algorithm. List each $\pi(s)$ starting with $s = 1$ to $s = n$ separated by a comma. That is, $\pi(1), \pi(2), \ldots, \pi(n)$. Note that since our actions are the outgoing edges, $\pi(s)$ should just give the vertex number of the outgoing edge that the policy returns.

Run your implementation on the two input files that are provided. Your submission must contain three parts: (1) submit the source code containing your implementation along with a README describing the compilation setup; (2) in the pdf report, write a brief description of your implementation. Report the number of iterations required for convergence. Explain the choice of the parameters used in your algorithm; (3) submit the two output files labeled `output_1.txt` and `output_2.txt`.

[1] Note that our reward function does not depend on the action you execute; instead it only depends on the current and future state. That is, $R(s, a, s') = R(s, s')$.