ECE 5984: Advanced Robot Motion Planning (Fall 2018)

Homework 2

Due: Thursday September 27th, 9PM

September 13, 2018

Instructions. This homework constitutes 10% of the course grade. You must work on it individually. 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 with no copy-pasting of significant portions of code from other sources.

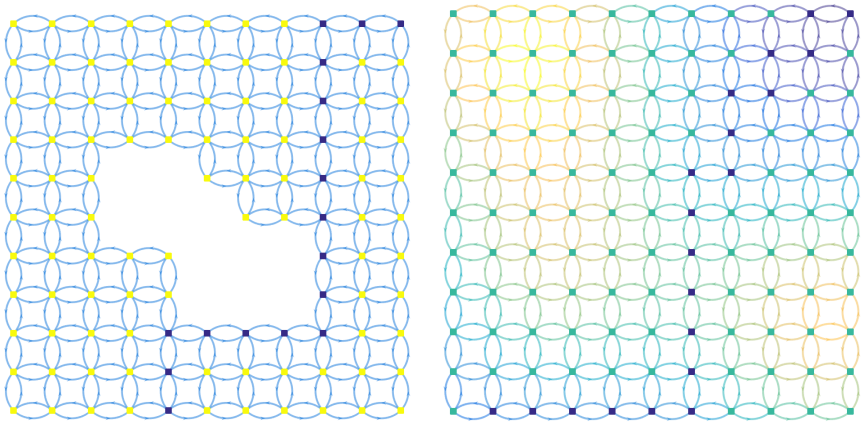
Please follow the submission instructions posted on canvas exactly. You must submit your assignment online on canvas by the due date. Your submission must include one pdf file with the answers to all the problems and one or more files containing your code for Problem 4. It is okay to scan your answers and create the pdf submission.

Problem 1 — 40 points

Implement the dynamic programming algorithm to solve the minimum cost path planning problem on a given graph. You may implement your algorithm using one of the following programming languages: MATLAB, C/C++, or Python.

Input Description. The input to your algorithm should be a file called `input.txt`. This file has a specific format as given below:

- 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.



(a) `input1.txt` (b) `input2.txt`

Figure 1: The two sample instances for Problem 1. The input is a directed graph given in the `input1.txt` and `input2.txt` files. The color of the edges indicates the cost (yellower edges have higher costs than bluer edges). The minimum cost path from start to goal is shown in blue.

Output Files. You may implement your algorithm using one of the following programming languages: MATLAB, C/C++, or Python. Your implementation must produce a single executable (for C/C++) or script (for MATLAB, Python) named `problem1` that reads in the input file and produces an output file called `output.txt`. This file must contain the vertices along the shortest path as well as the optimal value function (which is the first column in the table). The output file must have the following format:

- The first line must list the indices of the vertices on the shortest path – from the start to the goal vertex.
- The second line must list all the optimal value functions, i.e., $V(x_0), V(x_1), \ldots, V(x_n)$.

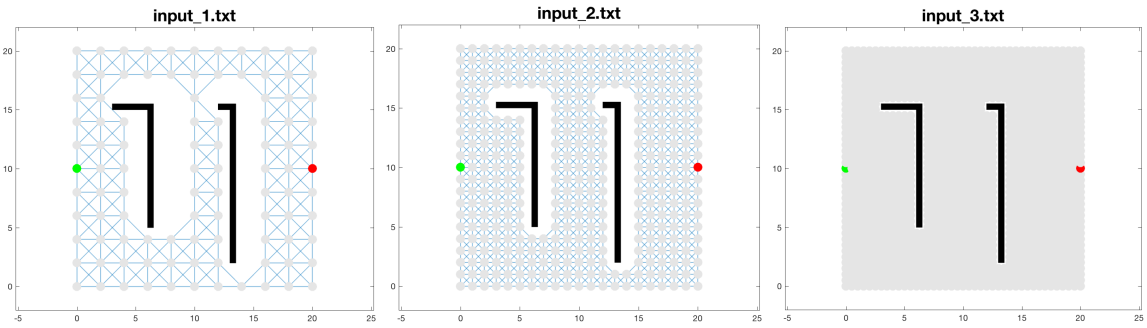
Two sample input and output files are given on canvas (also shown in Figure 1). The figures were drawn using the MATLAB graph function: `https://www.mathworks.com/help/matlab/ref/graph.plot.html`.

Report. In addition to the code, you must write a short paragraph describing your implementation in the pdf report. This description should include instructions on how to compile and run your code. We will test your code on instances other than the two input files. Make sure you follow the input/output conventions exactly.

Problem 2 — 40 points

Implement and compare the performance of Dijkstra and A* algorithms to solve for the minimum cost paths on a given graph. We will consider only 2D grid graphs in this assignment. The edge costs are the distances between the two vertices. Each vertex can be connected to at most eight of its neighbors (N,NW,W,SW,S,SE,E,NE).

Input Description. You are given three graphs that represent the same environment but with increasing grid resolutions (Figure 2). There are two input files corresponding to each graph: (1) the `input_*.txt` file specifies the input graph along with the start vertex and goal vertex in the same format as Problem 1; (2) the `coords_*.txt` file lists the x and y coordinates of the vertices starting with vertex numbered 1 on the first line. For example, the `input_1.txt` file contains 99 vertices. The `coords_1.txt` file specifies the actual coordinates of the 99 vertices starting with vertex numbered 1 to vertex numbered 99. The second input file will allow you to compute the heuristic (Euclidean distance to the goal vertex). You can also use this file to make it easier to visualize the paths produced by the algorithm. For example, in MATLAB R2016b you can use the



(a) `input_1.txt` (b) `input_2.txt` (c) `input_3.txt`

Figure 2: The three input files for Problem 2 represent the same grid environment with increasing grid resolutions. The start (green) and the goal (red) vertices are at the same grid position in all three graphs but have different vertex numbers since the graph sizes are different.

examples shown on this page `https://www.mathworks.com/help/matlab/ref/graph.plot.html#buzeikk` to display the graph. Similar plotting functions are available for C/C++ and python. See, for example, Graphviz `http://www.graphviz.org/Home.php`.

You do not need to visualize the paths or submit your visualizations. This is meant only to help you debug and understand what is happening in your code.

Output Files. You may implement your algorithm using one of the following programming languages: MATLAB, C/C++, or Python. Your implementation must produce a single executable (for C/C++) or script (for MATLAB, Python) named `problem2` that reads in all six input files (saved in the same directory as the executable or script) and produces two output files named `output_costs.txt` and `output_numiters.txt`.

Each output file must contain a 3×2 table with the first row (i.e., first line of the file) corresponding to `input_1.txt`, the second row for `input_2.txt`, and the third row for `input_3.txt`. Each line must list two numbers (i.e., two columns): for `output_costs.txt` these should give the cost of the paths returned by Dijkstra and A* with Euclidean distance as heuristic. For `output_numiters.txt`, the two numbers should correspond to the number of iterations taken by the two algorithm to find the output path (listed in the same order as the other output file). The number of iterations refer to the number of times we remove an entry from the closed list which is also the same as the size of the closed list at the end.

Your main executable or script must read in all input files, make specific calls to the Dijkstra and A* functions, and generate the two output files. Note that instead of writing two separate functions, you can just write one function which can be configured to run either algorithm.

Report. In addition to submitting code, you must write a short paragraph describing your implementation in the pdf report. This description should include instructions on how to compile and run your code. You must also include the two tables in the pdf report.

We will test your code on instances other than the input files supplied. Make sure you follow the input/output conventions exactly. Submit all files necessary to compile and run your code.

Problem 3 20 points

Gas Station Problem. Suppose that you have a robot that is tasked with traveling from start to goal. In this problem, we assume that we already know the path that will be followed by the robot. This path is a straight road from the start, x_0, to the goal x_n. Our robot has limited fuel and will need to stop along this path (possibly multiple times) to refuel. The robot spends one unit of fuel per unit distance traveled. There are n gas stations along the path: $x_0 \to x_1 \to x_2 \to \ldots \to x_n$. The i^{th} gas station, x_i, is at a distance d_i from the start. It costs p_i dollars per unit of fuel to refuel at the i^{th} station. Our objective is to minimize the total cost of refueling while ensuring the robot reaches the goal position.

(a) Can this problem be formulated and solved by dynamic programming?

(b) If your answer is yes, then write down the Bellman recurrence equation (similar to the one we saw in class) for this problem. Define and explain your notation clearly. You only need to specify the recurrence relation and not actually solve the DP. If your answer to part (a) is no, then give a counter example to show that the problem does not satisfy the optimal sub-structure property.

You may assume that initially the robot has B units of fuel. You can also assume that $B \geq d_{i+1} - d_i$ and that all d_i values are integers.