CSCI 360 – Project #1

Theoretical Part: A* - 2.5 Points

In this part of the project, you will compare two versions of A* with the same consistent h-values and the same pruning rule to not expand a node if a node labeled with the same state has already been expanded (but to consider the node expanded). Both versions always expand the unexpanded fringe node with the smallest f-value but one version of A* breaks ties among several unexpanded fringe nodes with the same (smallest) f-value in favor of the unexpanded fringe node with the smallest g-value and the other version breaks ties in favor of the unexpanded fringe node with the largest g-value. We want to figure out whether this kind of tie breaking makes a difference in the number of expanded nodes and, if so, which version of A* typically expands fewer nodes.

Characterize succinctly which nodes both versions of A* expand for sure (using their f-values and the goal distance of the start state) and which nodes none of the versions of A* expand. The remaining nodes might be expanded by one version of A* but not the other one.

Now consider an empty 4x4 grid with square cells, all of which are empty. A robot is always in exactly one cell and can always move north, east, south or west to a neighboring cell with cost one. The start cell is the upper left cell, and the goal cell is the lower right cell. Use the true goal distances of cells as consistent h-values and simulate (by hand) both versions of A*.

Which version of A* expands fewer nodes for this specific search problem than the other one?

Explain why this is so.

Is it the case for all search problems (that is, for all possible state spaces, start state, goal states and consistent h-values) that this version of A* expands no more nodes than the other one? Why or why not? (Try to be as precise as possible in assessing the merits of the version of A* that expands fewer nodes for the above search problem.)

Programming Part: Wheelbot - 2.5 Points

Moving an agent from a starting state or location to a goal state or location using local and global sensors is one of the most fundamental tasks in Artificial Intelligence. In this project, you will simulate a mobile robot performing this task on a discrete 2D grid using a simple text-based simulator. Below is a screen shot of the text-based simulator you will make use of in this course. Each dot represents a location in the environment that is currently not occupied. The 0 represents the location of the

robot in the environment, while the *$* represents the target location. The goal is to use the actions and sensors of the robot to move it from its current location to the goal location.

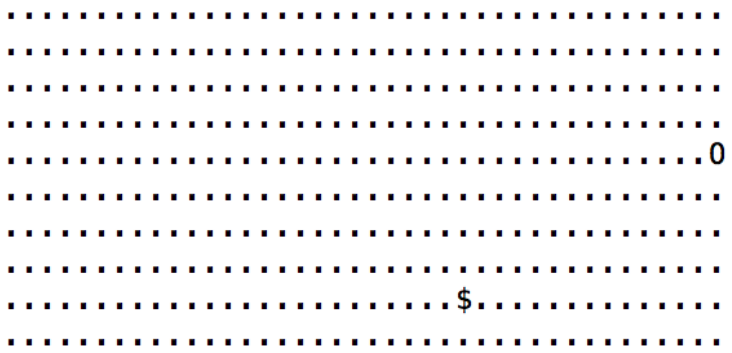

Figure 1: A screenshot of the text-based simulator. The environment is laid out in a discrete grid. Each period represents an unoccupied location in the environment. The robot's location is represented by the *0*, while the target location is represented by *$*.

Simulator Details

In this course, you will be working primarily with a robot called Wheelbot, an abstract mobile robot encoded in the *Robot* class. **At each simulation step, Wheelbot can read its sensors and then take one action, moving it one space in the given direction**. For the purposes of this assignment, Wheelbot has the following actions and sensors.

Wheelbot Actions

1. MOVE_UP: Wheelbot moves up one space in the grid
   (to use, call *r1→setRobotAction(MOVE_UP)*)
2. MOVE_DOWN: Wheelbot moves down one space in the grid
   (to use, call *r1→setRobotAction(MOVE_DOWN)*)
3. MOVE_LEFT: Wheelbot moves left one space in the grid
   (to use, call *r1→setRobotAction(MOVE_LEFT)*)
4. MOVE_RIGHT: Wheelbot moves right one space in the grid
   (to use, call *r1→setRobotAction(MOVE_RIGHT)*)
5. MOVE_UP_RIGHT: Wheelbot moves up and right diagonally one space in the grid
   (to use, call *r1→setRobotAction(MOVE_UP_RIGHT)*)

6. MOVE_UP_LEFT: Wheelbot moves up and left diagonally one space in the grid (to use, call *r1→setRobotAction(MOVE_UP_LEFT)*)

7. MOVE_DOWN_RIGHT: Wheelbot moves down and right diagonally one space in the grid
(to use, call *r1→setRobotAction(MOVE_DOWN_RIGHT)*)

8. MOVE_DOWN_LEFT: Wheelbot moves down and left diagonally one space in the grid
(to use, call *r1→setRobotAction(MOVE_DOWN_LEFT)*)

r1 is an instance of Wheelbot, and it will be provided for you to use in the code you need to modify.

Wheelbot Sensors

1. Robot Position Sensor: Returns the 2D position of the robot. To use this sensor, call *rob1→ getPosition()*, which returns a 2D vector. This is a local sensor of the Wheelbot.

2. Target Position Sensor: Returns the 2D position of the target. To use this sensor, call *sim1→ getTarget()*, which returns a 2D vector. This is a global sensor, which returns the absolute target location regardless of Wheelbot's position.

sim1 is an instance of the simulation environment and will be provided for you in the code you need to modify. The location of the robot and the target are returned as instances of the *Point2D* class that has the variables x (row) and y (column). The top-left corner of the map has coordinates (0,0).

Project Details/Requirements

This project will require you to modify the file Project1.cc in the project source code. **In this file, you will add code to the section of the file between TODO and END TODO, which is located in the *while* loop. You are not to modify anything else in this file or any other file that is part of the simulator.** Feel free to look at Robot.h, which defines Wheelbot, Simulator.h, which defines the environment, and Vector2D.h, which provides 2D vectors and points. We will test your project by copying your code block into our simulation environment in the designated area and running it. Any changes you make outside of the designated area will not be saved or counted as part of your submission.

The code that is between TODO and END TODO in the simulator package is placeholder code meant to illustrate how to control Wheelbot and use its sensors. This code should be removed and replaced with your own code. Your code must be standard C++ code. You will not need any headers, libraries, etc. outside of what is already provided in the Project1.cc file.

The requirements of the code you replace it with are as follows:

1. Use the above sensors and actions to navigate Wheelbot from its random starting position to the target position.

2. For full credit, your robot must take the minimum number of steps to the goal. Since the grid does not have any obstacles for this part of the project, you can easily figure out the shortest path without performing a search.

Submission

You should submit a PDF file named 'Part1.pdf' for your solution to the theoretical part and your modified Project1.cc file for the programming part through the blackboard system by Wednesday, February 10, 11:59pm. If you have any questions about the project you can post them on piazza (you can find the link on the course wiki) or come to the TAs' office hours.