Capstone Proposal

Aishwary Gupta
December 4th, 2017

Domain Background

The Robot Motion Planning problem as described in the projects list is a component of the Micromouse contest. This contest involves building and programming robots to most efficiently map a maze and then get to a certain destination in the fastest time. There are variants, and the rules can be changed to make the problem more complex or simpler. The robot needs to store some data about the maze as it does its exploration and do some processing to find a good route to its destination in the second run. This concept is essentially constant in the variations.

A problem like this should be solved and for more than just mental exercise. Mapping software is a good application of maze solving problem. The mapping software might not need to find its way through a maze, but just like a real robot in a maze, needs to figure out how to store directions and intersections as data, how to create a path between where the car is and where it needs to be, and the software also needs to decide what the shortest paths between destinations is. There is a lot of overlap in solving a maze and finding directions in the real world.

Problem Statement

In this particular instance of the problem, the robot has two runs to interact with the maze. Its score is added together as the number of actions it made in the second run and one thirtieth of the actions it made in the first run. An action is defined as an optional 90 degree rotation in either the left or right direction and an optional movement of up to 3 units forwards or backwards. The robot is limited to 1000 actions in either run. The robot sees the maze with 3 sensors, mounted on the front, left and right of the robot.

The maze itself is made of unit length walls, is solvable and has a 2x2 goal area in the center. The maze dimensions are 12x12, 14x14 or 16x16. The robot always starts at the bottom left corner facing up. It is presumed that the maze is closed, as in that the robot cannot physically escape the maze. The robot determines the number of action taken in the first run – at any time it can send an alert to end the run. The second run ends only when the robot enters the goal area.

Datasets and Inputs

The dataset is the set of maze instances we have. These are essential to the problem since without this data, the robot has nothing to do. The mazes themselves are all unique, but have interesting structures. The data was obtained from the problem specification itself. The data is very limited - while we can test out different robot algorithms, we would have to potentially have a random maze generator with certain specifications to do much else. Generating our own mazes is not a bad idea, especially to test how robust our robot is. We can even generate mazes larger than 16x16 to see how well our algorithms scale to larger sets.

Solutions Statement

Since the maze is perfectly defined in unit lengths, we can store the structure as a mathematical graph. The edges of the graph mark transitions between nodes and the nodes themselves are the tiles the robot could be on. The nodes could have interesting properties such as how much the robot knows about them, distance from center, etc. The robot can then use graph algorithms to potentially determine its next direction, and other maze solving routines. The robot creates the graph as it explores more of the maze.

In the first run, the robot explores the maze, building the graph, In the second, it uses the graph to find the shortest path to the center.

Benchmark Model

The benchmark is provided in the problem specification as a score. The lower the score, the better it is since it means the robot took less time to achieve the goal. Since there is one maze for each of the three dimension types, it may be difficult to compare benchmarks across the mazes since the mazes and their dimensions are different. But, this doesn't stop us from benchmarking different algorithms to each other. The algorithm scalability can also be explored across the three mazes - for instance algorithm A could work very well in a 12x12 maze but do poorly on a 16x16. The robot could take this into consideration when determining what to do, if the robot has a set of algorithms to choose from. Since the benchmark itself is a numerical score, it is easily measured.

A second option for scoring is to consider a maze complection metric - a value that describes how much the robot knows about the maze in general. The more a robot learns per move, the better.

Evaluation Metrics

The percentage completion of the maze is likely a good evaluation metric. It is more descriptive than the provided scoring from the tester.py module. This is because the score from tester.py incorporates both runs together, while we are more generally interested in the first run. The number of steps in the second run depend entirely on the quality of data gathered during the first run. The more we learn about a maze per action the better. Hence, our evaluation metric will be a percentage completion value. Since there are 4 sides to each tile, we'll add up how many of the sides the robot knows for each tile and then divide that number by four times the number of tiles that exist in the maze.

This percentage is very useful in that it can identify pitfalls in algorithms such as when they seem to be wasting moves and the algo can also help us see where our robot is doing well and what circumstances make it possible for our robot to learn the most with each time step.

Project Design

The main focus will be crafting a strong exploration algorithm. The data the robot collects will be stored in a graph, since it is flexible and robust. The robot will have some way of deciding when it is done exploring the maze, or feels it has learned enough of the maze to commence the second run. On the second run, the robot must have a path to the center to even get there. It can calculate the shortest path using Dijkstra's algorithm.

During the exploration segment, the robot will use its sensors to see squares in its surroundings. If it sees two ahead, it knows things about those tiles, and the tile on the other side of the wall. While it may not be able to connect the tile on the other side of teh wall to the graph, it still knows something about that tile and can fill in the information on it later. The robot needs to decide which way to move to explore the maze. It can examine the different branches of the graph to see the most promising locales and venture there, or it could explore a node immediately next to it as a greedy approach. The point is that there are a lot of heuristics the robot could take to fill in the graph that is the map of the maze.

The robot needs to make the decision to end the first run. Knowing when to make this decision is very important. Done too soon, and the robot won't know the shortest path. Done too late, and the robot may have wasted time on gathering useless information. Also, it is important to mention that this is very applicable to the real world - a mapping algorithm should be smart enough to know when further computation won't yeild any useful results - the world is too big anways to explore all of it to find the fastest path between two points.

Once the exploration is done, the second run is straightforward - Dijkstra's is guaranteed to find the shortest path for a given graph, and the robot then takes it.