Machine Learning Engineer Nanodegree

Capstone Proposal

Mayank Indoria
September 24th, 2017

Proposal: Robot Motion Planning

Domain Background

The project is derived from the *Micromouse competitions* [1] in which a small robot tries to solve a 16x16 maze. The robot starts from a corner and the goal is to reach the central area of the maze. The maze environment is completely unknown to the robot. The robot first needs to explore and map the unfamiliar maze environment localizing itself in the process. While exploring, the robot needs to detect whether the goal is reached at least once. After exploration, the robot needs to find an optimal route from the start state to the goal state using the information gained during the exploration phase. The robot is given multiple runs, in the first run the robot explores the maze and find paths from start to the goal and in the following runs the robot tries to reach the goal in shortest time possible, selecting one of the paths which is optimal.
The goal of this project is to devise a strategy that a robot will use to navigate a virtual maze, and find an optimal path to reach goal from the starting position. A path is optimal if it takes shortest time to reach the goal.

Problem Statement

Maze (M):

The maze (M) is an $n \times n$ grid consisting of square cells. n can take values that are 12, 14 or 16. There are structures called walls that block the robot motion, a robot can not go past a wall. The grid is completely surrounded by walls on the outside boundary. Walls can also be found on the connecting edges of the internal cells.

Start (S):

The square cell at the bottom left corner of the maze (M) is the starting position of robot. Always, the right side of the starting cell is blocked by a wall.

Goal (G):

The *2 x 2* square area located at the center of the grid is the goal.

Robot (R):

The robot, at any point of time, occupies a square cell of the maze grid (M). It always lies at the center of occupied cell. The robot is equipped with three sensors, located at the front, left and right side of the robot. The respective sensors (at front, left and right side) provides information about open squares cells in front, left and right side of the robot.(Number of cells in a particular direction until a wall is encountered are open square cells). At any step, the robot can turn (clockwise or anticlockwise) by ninety degrees and can move up to three steps forward or backward. The sensors and robot's motion is assumed to be perfect.

Path (P):

A series of cells, $p = (c_1 , c_2 , \ldots\ldots , c_n)$ - is called a path, p, from c_1 *to* c_n *if* for every c_i *and* c_j *in p,* c_i *and* c_j are adjacent (4-adjacency) to each other without being separated by a wall.

Exploratory Run:

First run of the robot (R) on a maze (M). During exploratory run the robot (R) navigates the maze (M) and builds an internal representation of the maze called the map.

Exploitatory Run:

Second run of the robot (R) on a maze (M). During this phase the robot uses map of the maze to plan a path from start (S) to Goal (G)

Problem :

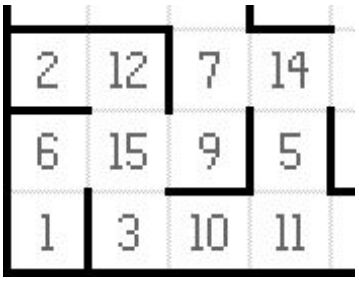
A robot (R) in a maze (M) (unknown to the robot) placed at starting position S and given the destination goal G,
The robot, using the exploratory run, needs to find a path p = $(c_1 , c_2 , \ldots\ldots , c_n)$ **where** c_1 **=** ***S*** ***and*** c_n **=** ***G*** **such that the path** ***p*** **is optimal. A path** ***p*** **is optimal if it takes shortest time to arrive at goal** ***G*** **from the start** ***S*****.**

Datasets and Inputs

A maze is provided as input using a text file. The first line of the text file is a number *n* indicating the dimension of the maze grid. A *n x n* maze contains n^2 square cells. The first line is followed by *n* lines. Each of the *n* lines contains *n* comma seperated values that describes which edges of a square cell are blocked by a wall. The first of the *n* lines corresponds to the leftmost column of the maze starting from the bottom and so on. Each comma separated value is a 4-bit binary number. A ***1*** at a bit position indicates an open edge and a ***0*** at a bit position indicates a closed edge (blocked by a wall). Each bit position corresponds to one of the four edges of a square cell and the correspondence is as follows:

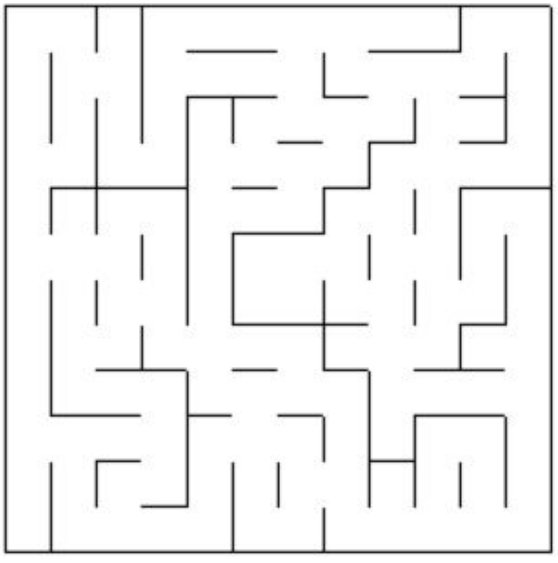
Bit position 0 (LSB, weight = 1) : Upward edge
Bit position 1 (weight = 2) : Right edge
Bit position 2 (weight = 4) : Bottom edge
Bit position 3 (MSB, weight = 8) : Left edge

An Example input with the corresponding maze:



A cell which has edges to its left and right side corresponds to the value:
1*1+0*2+1*4+0*8 = 5, that can be seen in the sample maze.

Three test mazes are provided for testing and evaluating the solution. One of the test maze is shown on the left.



Solution Statement

The problem has two phases - Exploratory run and Exploitatory run.

The maze is unknown to the robot. To plan any path from start to goal position, the robot needs information about the maze. During Exploratory run the robot navigates the maze and build an internal representation (map) of the maze. Map of the maze is built as a Graph *H.* The vertices of the graph *H* are the cells of the maze. During exploration phase the edges are introduced between the vertices of *H.* An edge *e* is introduced between two vertices v_i *and* v_j if :

1. v_i *and* v_j are adjacent cells (4-adjacency)
2. v_i *is reachable from* v_j , that is there is no wall separating the two vertices.

The robot at any point of time, occupies a cell in the maze. The robot uses sensor data to find reachable cells from the current cell in three directions (front, left and right of the robot). It then stores the reachable cells and chooses one of them (not visited earlier) to move to and repeat this process and in this way it expands the graph *H.* A graph search algorithm will be used in for expanding the graph *H* (which differs in the way the next vertex is chosen for expansion) like Breadth First Search (BFS), Depth First Search (DFS) or Best First Search. During this phase the robot should visit the goal *G* at least once, that will make sure that there is a path from start to goal.

Exploratory phase gives a graph *H* as output. In the second phase which is Exploitatory phase the graph *H* is used to find an optimal path from the starting cell to the goal. There may exist multiple paths from start to goal, the shortest path finding algorithm at this phase finds the best path to arrive at the goal.

The solution will be quantified as a path $p = (v_1 , v_2 , \ldots\ldots , v_n)$ *where* $v_1 = S$ *and* $v_n = G.$ The length of a path, *p*, will be measured as the number of vertices required to reach the goal from the starting cell.
Given that the graph *H* has the optimal path (the optimal path is explored during first phase), the shortest path finding algorithm will always produce the same optimal path (or more than one optimal paths if exists).

Benchmark Model

Two benchmarks will be considered, and each benchmark will be evaluated using the metric defined in the following section, that is:

$$\textbf{score} = steps_{path} + (steps_{exploration} / 30)$$

where, $steps_{exploration}$ **:** Number of steps taken by the robot during exploratory run
$steps_{path}$ **:** Number of steps taken by robot to reach the goal from the start following the path planned for second run.
A lesser score means a better solution to the problem.

Two benchmarks score would be considered for every maze : *benchmark-1* and *benchmark-2*. The *benchmark-1* score would be the target that a robot would try to beat or come as close to it as possible .The *benchmark-2* score would be the lower limit of performance and any solution needs to do better than this.

For *benchmark-1*,

$$steps_{path} = steps_{optimal\text{-}path}$$

where $steps_{optimal\text{-}path}$ *is the length of the optimal(smallest number of steps) path that exist between start and goal for a particular maze.*

For a robot in any maze, to find the optimal path that exist to reach the goal area, it has to visit every grid cell at least once and it will take a very well thought exploration strategy to find an optimal path without visiting each cell once. So for this benchmark

$$steps_{exploration} = n^2 \text{ , where n x n is the maze dimension}$$

$$benchmark\text{-}1 = steps_{optimal\text{-}path} + n^2/30$$

For *benchmark-2,* a robot ,which during exploratory phase, randomly chooses a cell to move to will be considered. Such a robot might not reach the goal each time. So, the exploratory runs where the robot reaches the goal will be considered. Once the robot reaches the goal the exploratory phase will end. The average score of 10 such trials would be considered as *benchmark-2.*

$$benchmark\text{-}2 = \Sigma_i (steps^i_{path} + (steps^i_{exploration} / 30))/10$$

where, $1 \leq i \leq 10$

benchmark-1 is a hard to break threshold, any solution that *scores* less than this benchmark or comes close to this benchmark will represent a quality solution. So, *benchmark-1* will be the target that any solution will try to meet. On the other hand, *benchmark-2* represents worst score that can be achieved on a given maze. Any solution to the problem must *score* less than this value for a particular maze.

Evaluation Metric

The robot is scored on every test maze. The score is given on both the runs. During the first run the robot explores the map to expand its knowledge about the maze. Even if the goal is reached the robot can continue exploring the maze, but once the goal is reached the robot can end its run at any point of time. In the second run, the robot plans an optimal path to arrive at the goal using the map built during exploration and follows the optimal route to the goal.

Let,

$steps_{exploration}$ *:* Number of steps taken by the robot during exploratory run

$steps_{path}$ **:** Number of steps taken by robot to reach the goal from the start following the path planned for second run.

Then the score given to the robot for a maze is,

score = $steps_{path}$ + ($steps_{exploration}$ / 30)

Hence, the score is sum of the length of the optimal path and one thirtieth of the steps taken by robot during exploratory phase. The smaller the score is, the better is the performance for a particular maze. So, the aim is to minimize the score.
For a single maze, maximum of thousand steps are given to complete both the runs.

The robot incurs penalty in both the runs. The robot is penalised if it is unable to find an optimal path and also if the robot spends much time in exploring the maze (using some inefficient exploration strategy). So, it is possible that even after finding an optimal path a robot that spends much time in exploring the maze performs worse than a robot which finds a suboptimal path but takes less time steps in exploration phase.

Project Design

- **Languages and Libraries**
    - Python 3.5.x
    - Numpy

- **Workflow**
    - The project is provided with a starter code that includes the following files
        - **`robot.py`**
        - **`maze.py`**
        - **`tester.py`**
        - **`showmaze.py`**
    - First step would be understanding the code provided, what is the functionality of each file and how the code in different files interact with one another.
    - Exploring the test mazes. Three test mazes are provided in the files **`test_maze_##.txt`**. These are the mazes on which a solution will be evaluated. It is important to explore the mazes and discover some structural observations which will be important while designing a solution to the problem.
    - Different algorithms can explore the maze, each differing in the way the next cell is chosen to move
        - **Random movement:** In this strategy at any particular time step, the robot will choose its action randomly. That is the choice of the next cell the robot will move to is made randomly.

    - **Breadth First Search(BFS)**[2]**:** In this algorithm, the cells are visited in order of their discovery, that is in *First In First Out (FIFO)* manner. When at a particular cell, all the unvisited neighbouring reachable cells are added to a Queue that stores the cells that are discovered but not yet visited. In BFS, all the cells are visited in order of their distance from the starting cell.
    - **Depth First Search(DFS)** [3]**:** Thce cells are visited in *Last In First Out (LIFO)* order. The latest discovered cell will be visited next. So, the robot will follow a path in the maze, as deep as it goes, until a dead end is encountered or goal state is reached, or a previously visited cell is encountered.
    - **Best First Search**[4]**:** The above three strategies chooses the next cells to visit blindly without considering whether to visit a cell would be a good or a bad decision. Best First search uses a function (called a *heuristic*) to decide which cell to visit next. *Heuristic* function provides the most promising node to visit among a set of nodes. An example of Best First search algorithm is **A*** search algorithm[5].
- All kind of algorithms stated above can find an optimal path to the goal, but they can be used more intelligently in the given scenario of maze search. The sensors of the robot provide information about more than one open cell in a particular direction and the robot can move 3 steps at a time. The robot is not required to visit each cell to form a map of unknown areas of the maze and thus accelerates the process of exploration. So, the above algorithms should be modified to use the information provided by sensors.
- The *score* of the robot on a particular maze can also be improved by keeping track whether an optimal path to the goal has been discovered. If the maze is not fully explored yet and and an optimal path is discovered, the robot could end its exploratory run and hence can improve upon the *score.* If it is known that a path to the goal is already found (the robot has reached the goal at least once) and exploring further can not contribute in making path more optimal, then one of the path in the set of the paths found is optimal path. This strategy can be used to end the exploratory run of the robot.
- Implement solutions, run on each test maze and record the result. Each solution will be evaluated on the basis of the scoring metric defined in an earlier section.
- Analyze the best solution (which incurred minimum scores), and provide scenario where the solution can get stuck if any, providing poor results. Refine the algorithm to tackle such a scenario.
- Comparing the final solution with the benchmark and provide justification why the final solution is appropriate.

References

1. Micromouse. (2017, June 29). In *Wikipedia, The Free Encyclopedia*. Retrieved 20:13, September 23, 2017 from https://en.wikipedia.org/w/index.php?title=Micromouse&oldid=788131827

2. Breadth-first search. (2017, September 19). In *Wikipedia, The Free Encyclopedia*. Retrieved 20:18, September 23, 2017, from https://en.wikipedia.org/w/index.php?title=Breadth-first_search&oldid=801398840

3. Depth-first search. (2017, August 28). In *Wikipedia, The Free Encyclopedia*. Retrieved 20:21, September 23, 2017, from https://en.wikipedia.org/w/index.php?title=Depth-first_search&oldid=797704202

4. Best-first search. (2017, January 7). In *Wikipedia, The Free Encyclopedia*. Retrieved 20:21, September 23, 2017, from https://en.wikipedia.org/w/index.php?title=Best-first_search&oldid=758785744

5. A* search algorithm, http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

6. MLND capstone project description- Robot Motion Planning, https://docs.google.com/document/d/1ZFCH6jS3A5At7_v5IUM5OpAXJYiutFuSIjTzV_E-vdE/pub