Machine Learning Engineer Nanodegree

Capstone Proposal

Yang Hao
April 15st, 2017

Proposal

Domain Background

This project in which we are working consists in programming a virtual robot to explore a maze from a predefined corner to its center, and finding the best way to reach this center. There are several path to reach the goal, among them of course, the robot should learn the shortest one. This project takes inspiration from Micromouse competitions, wherein a robot mouse is tasked with plotting a path from a corner of the maze to its center. The robot mouse may make multiple runs in a given maze. In the first run, the robot mouse tries to map out the maze to not only find the center, but also figure out the best paths to the center. In subsequent runs, the robot mouse attempts to reach the center in the fastest time possible, using what it has previously learned. The goal of this project is to define and implement a strategy to consistently discover an optimal path through a series of test mazes that exist within a virtual world inspired by the Micromouse problem.

Problem Statement

The problem to solve is : having an 12x12 or 14x14 or 16x16 maze where the goal is placed in the center, run this in the shortest possible time from the origin to the goal. The maze exists on a grid of n squares in column and n squares in rows (*n* even). The minimum value of *n* is twelve, the maximum sixteen. Along the outside perimeter of the grid, and on the edges connecting some of the internal squares, are walls that block all movement. The robot will start in the square in the bottom- left corner of the grid, facing upwards. The starting square will always have a wall on its right side (in addition to the outside walls on the left and bottom) and an opening on its top side. In the center of the grid is the goal room consisting of a 2 x 2 square; the robot must make it here from its starting square in order to register a successful run of the maze. So to do this, the robot has to perform a first run of the maze in order to build a knowledge of it. This first run cannot end without finding the goal. After entering the goal room, the robot

may choose to end its exploration at any time. The robot is then moved back to the starting position and orientation for its second run. Its objective now is to go from the start position to the goal room in the fastest time possible. The robot's score for the maze is equal to:

- the number of steps required to execute the second run,
- plus the number of time steps required to execute the first run divided by one thirtieth.

A maximum of 1000 time steps are allotted to complete both runs for a single maze.

Datasets and Inputs

Mazes are provided to the system via text file. On the first line of the text file is a number describing the number of squares on each dimension of the maze *n*. On the following *n* lines, there will be *n* comma-delimited numbers describing which edges of the square are open to movement. Each number represents a four-bit number that has a bit value of 0 if an edge is closed (walled) and 1 if an edge is open (no wall); the 1s register corresponds with the upwards-facing side, the 2s register the right side, the 4s register the bottom side, and the 8s register the left side. For example, the number 10 means that a square is open on the left and right, with walls on top and bottom (0*1 + 1*2 + 0*4 + 1*8 = 10). Also, the maze that the robot should run must be randomly generated in order to let us be able to test our solution in several input data.
Certain mazes are not solvable, i.e there are no ways to reach the goal. So there should be a tester algorithm to verify it when given a maze.

Solution Statement

What we are going to implement with the help of algorithms and techniques discussed above, is to **store and update the robot knowledge of the maze**, to **determine how far the robot should move and in which direction to maximise knowledge of the maze while exploring yet minimise the number of steps taken during exploration**, to **find the goal location and make sure the Robot has visited it** (this is a requirement to allow the robot to end the exploration run early), to **determine if the path to the goal is optimal**, once the optimal path has been found, save it, end the exploration run, and **execute the second run up to the goal.**

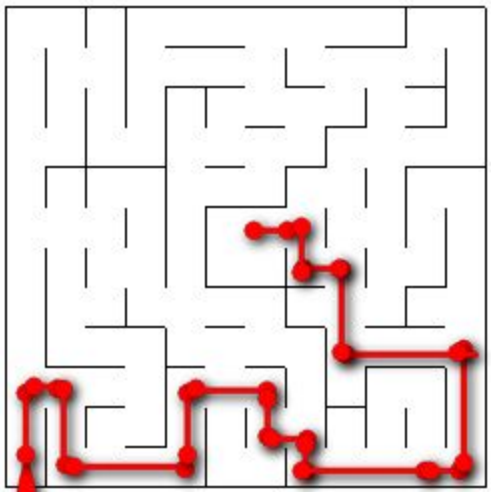
Benchmark Model

The performance criteria of this problem is based on the score. What we call score of the robot is formulated as follows : ***score = n_step2 + n_step1/30***.

Our benchmark will be based on this formula. The parameter n_step2 is the number of time steps required by the robot to reach the goal. It depends on the optimal path that have been previously calculated by the robot in the first run. So for a particular maze, best path will always be the same whatever the number of time that the robot executes the second run.
The parameter n_step1 is the number of step to execute the first run. This parameter can vary according to the chosen algorithm. So it the keystone of our benchmark model.

Evaluation Metrics

For the test maze one, the lower n_step1 is 9.6 while the best is 3.667. For the same test maze one, n_score2 = 17. Our solution should be between those two value. Notice that the score is the smaller one and not the higher. Our solution will be acceptable if, for the maze one, the score is close to 17+3.667 = 20.667. It will not be a good solution if the score is close to 17+9.6=26.6.



Here is the solution for the test maze one.

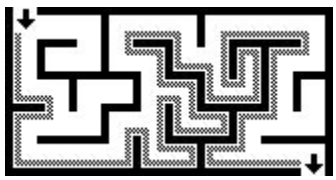
Project Design

There are many algorithms designed to solve the navigation of a maze using the optimal path. Among them we have: Random mouth algorithm, Wall follower algorithm, pledge algorithm, A* algorithm, minimax, alpha beta pruning, Dijkstra, The A*, Dijkstra's, Graph traversal algorithms. This problems will best be solved using reinforcement learning techniques. All this techniques work on a graph. This implies that the maze must first be modeled by a graph. In the following, we are going to explain what

- **Dijkstra's algorithm**: helps finding the shortest or optimal paths from a source vertex v to all other vertices in the graph. When the robot is running the maze for the first time (exploration), he builds a graph at the same time. After the graph is

built, for the second run we can use Dijkstra's algorithm to compute the optimal path to the center.

- **Graph traversal algorithms**: Those are algorithms for exploring graphs starting at one of its vertices, i, and finding all vertices that are reachable from i. The maze can be modelized by a graph where vertices are squares. In the exploration phase, the robot goes from the start vertex *i0* and finds all vertices that reachable from i0 in order to *i0*. There exists two variants of this algorithms :
    - ***Depth-first-search*** visits the child vertices before visiting the sibling vertices. It means, it traverses the depth of particular vertices before exploring its breadth. This algorithm takes a Graph G and vertices v as inputs and returns a labeled graph G' from G.
    - ***Breadth-first search:*** The simplest of the graph search algorithm, it visits the neighbour vertices before visiting the child vertices. They often use it to find the shortest path from a vertex to another
- **Tree traversal algorithms**: Algorithms to access node of a tree. It is special case of graph traversal. A tree is a particular graph while a graph is not necessarily a tree.
- ***A* algorithm*** : is a modification of Dijkstra's Algorithm that is optimized for a single destination. Dijkstra's Algorithm can find paths to all locations; A* finds paths to one location. It prioritizes paths that seem to be leading closer to the goal. Dijkstra's Algorithm works well to find the shortest path, but it wastes time exploring in directions that aren't promising. Greedy Best First Search explores in promising directions but it may not find the shortest path. The A* algorithm uses *both* the actual distance from the start and the estimated distance to the goal.
- **Wall follower algorithm** :



The robot runs following the walls.