Robot motion planning

Ilya Semenov

14 May 2017

Plot and Navigate a Virtual Maze

Capstone project proposal for Udacity machine learning engineer nanodegree

Project description

The 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 two 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. During the second run, the robot mouse attempts to reach the center in the fastest time possible, using what it has previously learned.

Maze solving is a well known academic problem in the robotics field, some approaches described, for example, in following papers:

- Maze Solving Algorithms for Micro Mouse
- An Efficient Algorithm for Robot Maze-Solving
- A New Shortest First Path Finding Algorithm for a Maze Solving Robot

Problem statement

Maze definition

The maze exists on an N x N grid of squares, N is 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.

Robot definition

The robot can be considered to rest in the center of the square it is currently located in, and points in one of the cardinal directions of the maze. The robot has three obstacle sensors, mounted on the front of the robot, its right side, and its left side. Obstacle sensors detect the number of open squares in the direction of the sensor; for example, in its starting position, the robot's left and right sensors will state that there are no open squares in those directions and at least one square towards its front. On each time step of the simulation, the robot may choose to rotate clockwise or counterclockwise ninety degrees, then move forwards or backwards a distance of up to three units. It is assumed that the robot's turning and movement is perfect. If the robot tries to move into a wall, the robot stays where it is. After movement, one time step has passed, and the sensors return readings for the open squares in the robot's new location and/or orientation to start the next time unit.

Complement to Udacity robot definition

- sensor range in all directions is three cells for all three sensors
- robot recieves sensor data only when it reaches a final cell of movement sequence (for example, if robot moves two cells forward it recieves sensor data from the second cell only)

Note

- according to robot definition from above diagonal moves are forbidden

Scoring

On each maze, the robot must complete two runs. In the first run, the robot is allowed to freely roam the maze to build a map of the maze. It must enter the goal room at some point during its exploration, but is free to continue exploring the maze after 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. The robot's score for the maze is equal to the number of time steps required to execute the second run, plus one thirtieth the number of time steps required to execute the first run. A maximum of one thousand time steps are allotted to complete both runs for a single maze.

Robot's goal

Using the maze configuration data collected during explorational phase move from start position to the center of the maze as quickly as possible during the final phase.

Inputs

Maze configuration

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 ($0*1* + *1*2 + 0*4* + *1*8 = 10$). Note that, due to array indexing, the first data row in the text file corresponds with the leftmost column in the maze, its first element being the starting square (bottom-left) corner of the maze.

Robot interface

At the start of a time step the robot will receive sensor readings as a list of three numbers indicating the number of open squares in front of the left, center, and right sensors (in that order) to its **next_move()** function. The **next_move()** function must then return two values indicating the robot's rotation and movement on that timestep. Rotation is expected to be an integer taking one of three values: -90, 90, or 0, indicating a counterclockwise, clockwise, or no rotation, respectively. Movement follows rotation, and is expected to be an integer in the range [-3, 3] inclusive. The robot will attempt to move that many squares forward (positive) or backwards (negative), stopping movement if it encounters a wall.

Starter code

Robot and maze specifications along with other support code provided by Udacity will be downloaded from the following link.

Data preprocessing

No data preprocessing required in a given project pipeline.

Proposed solution

My general approach is to build a maze connectivity graph during an exploration phase and on each step of exploration find an optimal solution for travelling from start to goal cells. Since each cell has a unique address that could be represented by an integer (for example from 1 to 256 in 16x16 maze case) we can

define graph vertices as cells and define graph edges as number of time-steps required to travel from one cell to another.

Initially the connectivity graph has no edges. Each exploration step adds edges to the graph according to the data obtained from robot sensors.

For optimal solution search we can use Dijkstra's algorithm since it has complexity upper bound O(V-squared), where V is the number of graph vertices. According to problem statement maximum number of vertices is limited to 256 (in case of 16*16 maze grid), so performing Dijkstra optimal path search on each step of exploration phase looks as feasible task in terms of computational efficiency.

In this framework final phase is just an implementation of optimal path found during exploration in case it exists and was found with boundary condition of 1000 time-steps.

Exploration phase may be defined as reinforcement learning task (similar to Train a smartcab to drive exercise in the course) with strategy described by some heuristics dedicated to minimize time required for optimal path search.

Exploration heuristics

- robot recieves award for moves towards the maze center
- robot is penalized for moving to a previously visited cell
- exploration stops immediately if a path with time-length of 15% or less of total number of cells is found
- robot is penalized for vertical moves towards the edges of the maze

Exact values for awards and penalties are to be defined during the implementation of the algorithm.

Model benchmark and evaluation metric

Model perfomance on any given maze can be scored as average number of times-steps required to find and follow a path with arbitrary number of attempts (by now 10 attempts seems as enough). We need several attempts for given maze configuration due to probabilistic nature of reinforcement learning algorithm.

This benchmark allows comparison between different models on a given maze configuration. To be more clear we'll explain the benchmark with an example. In a table below each row represents (imaginary) total number of time-steps required for completion both explorational and final phases.

No of test run	Model 1	Model 2
1	100	95
2	100	103
3	100	101
4	100	97
5	100	111
6	100	91
7	100	92
8	100	93
9	100	89
10	100	93
Average time-steps required	**100**	**96.5**

The benchmark for the Model 1 is 100, the benchmark for the Model 2 is 96.5. Since Model 2 has better benchmark than Model 1, one can claim that Model 2 coped better with a given maze configuration.

In order to evaluate model perfomance we can also compare the benchmark with lower and upper bounds for optimal path search on a particular maze. It could be useful to judge whether a model needs further optimization.

Lower bound defined as a shortest path length (expressed in the number of time-steps) in a given maze.

Upper bound defined as 1/30 of time-steps needed to build a complete connectivity graph by visiting all the available cells for a given maze plus the lower bound. There is some ambiguity in upper bound definition since for most maze configurations there are several ways to visit every cell. In order to avoid the ambiguity my proposal is to implement an analogue of the depth-first-search in a graph where the robot is allowed to move no farther then exactly one cell in any direction during the exploration and, where possible, prefers left turns to forward moves and forward moves to right turns subsequently.

The difference between upper and lower bounds gives a scale where benchmark described above can be expressed in % thus giving a metric to judge whether the model performance closer to ideal optimal path defined by lower bound or slow maze exploration by visiting every cell, that is upper bound. The example of that metric is as follows:

Let in an arbitrary maze optimal path length consists of 10 time-steps and slow maze exploration with visiting every available cell takes 260 time-steps. In that case:

- Lower bound equals 10.33 (10 time-steps divided by 30 for an exploration phase and 10 time-steps for a final run)
- Upper bound equals 18.67 (260/30 for exploration phase plus 10 time-steps for a final run)

Let the model benchmark evaluated with benchmark algorithm described above equals 111.6. Than the metric will be evaluated as follows:

(111.6 - 10)/30 + 10 = 13.37

percent_metric = ((13.37 - 10.33) / (18.67 - 10.33)) * 100% = 36.5%

The value of a metric can provide some insights on how the model performance is far from an ideal pathfinding behaviour. The ideal value of a metric is 0%.

Note

In case if a robot choose to follow non-optimal path according to learning heuristics the value of a metric could be more than 100%.

Project design

A robot carries 'onboard' a maze connectivity graph that represents path structure of a maze. Vertices of the graph correspond to maze cells whereas edges correspond to passages between neighboring cells. Edges have weights corresponding with time to reach appropriate cell from current cell. That allows to take into consideration time robot needs to make a turn left or right. Also, according to suggested project pipeline, the project is an implementation of next_move() function in the *robot.py* file.

Graph initialization procedure

Connectivity graph 'lives' in a separate Graph class. Instance of this class is being created at the beginning of maze exploration phase inside ___init() function of *robot.py*. Since we know maze dimension at this step we can fill graph vertices set and connect four central vertices (target cells) with four edges, since target cells are always a square of four cells without walls between them by problem formulation.

Inner implementation of the connectivity graph is adjucency list since this is more appropriate for Dijkstra algorithm in terms of memory consumption and running time.

Since we have initial maze orientation, we can unequivocally number vertices from 0 by rows from left to right, where vertice No 1 corresponds to starting cell. Also we can always calculate the vertices numbers for target center cells using given maze dimension.

The result of graph initialization procedure is an extremely sparse graph with No of vertices corresponding to maze dimension (for example, 144 vertices for 12x12 maze) and only four edges between central cells.

In this framework the goal of exploration phase is filling connectivity graph with available edges trying to make the passage to central cells as fast as possible.

Robot movement rules

During exploration phase robot movement defined by maze exploration strategy.

During final run robot movement defined by shortest path towards the maze center calculated with Dijkstra algorithm on connectivity graph from current vertice. If no path found in the connectivity graph robot don't move at all.

Maze exploration strategy

The following strategy applied consequently during each step of an exploration phase.

- given the sensors input robot adds edges to connectivity graph from current vertice to neighboring vertices
- robot runs Dijkstra's algorithm on a connectivity graph from the starting point to any of the goal vertices (we can arbitrarily choose any vertice since they are connected during graph initialization). If such path is found and its length is 15% or less of total vertices number (see *exploration heuristics* described above) **next_move()** returns ('Reset', 'Reset') tuple thus haulting exploration phase
- if the path is found but its length does not satisfy the condition above, robot haults further exploration with probability of one half of total steps made by the moment divided by 1000 (total time steps allowed for run). This condition partially prevents the robot from further ineffective maze space exploration when some path, probably not optimal, already found. 'One half' coefficient for this probability may be corrected during my final perfomance tests of the model
- robot maintains a list of vertices, that are not visited yet, but available for the exploration according to sensors input obtained so far
- every maze cell belongs to a virtual layer, where target central cells belong to layer 0, all cells that surround central cells belong to layer 1, and so on up till the outer layer, that contains the starting cell. Total number of layers equals half of a maze dimension. The idea of layers gives a way to stimulate moves that lead to the center of the maze and penalize moves towards outer borders
- final stage of a strategy is a choice of next cell that robot is intended to move into. It has to choose one point from a tuple of at maximum 12 destination points. Every point in a tuple has a score that represents the value of moving into this point. Tuple points scoring requires two steps
- first step. Each available point scored from -3 to 3 based on how many layers this point is closer to the center from the current robot position (we'll call that number 'layer difference'). Than every point marked as 'visited' recieves additional -3 penalty.
- second step. Robot traverses 'not visited' list and calculates shortest path from the current point to every point from this list. Any tuple point lying on that path recieves added score that equals layer difference divided by path length in time-steps.
- final move selection. If a tuple with weighted points obtained by the above procedure contains points with positive weights, robot throws away all points with non-positive weights and chooses randomly next point from remaining positive-weighted with probability normalized by point weight. If there are no positive weights in a tuple, all weights are increased by largest absolute weight and robot makes random selection based on a point probability given by normalized weight.

The goal of a sophisticated next move selection procedure described above is to implement robot's behaviour similar to depth-first-search graph exploration that is focused to reveal available paths leading as straight to the maze center as possible.

Capstone project scope

The scope of a Capstone project is to implement the algorithm described above and tune the exact values of model coefficients in order to get satisfactory benchmarks on different test mazes.

References used in a document

- Micromouse competition description
- Udacity project pipeline
- Dijkstra's algorithm (with running time estimate)
- Maze Solving Algorithms for Micro Mouse
- An Efficient Algorithm for Robot Maze-Solving
- A New Shortest First Path Finding Algorithm for a Maze Solving Robot