9/23/2017

The Delivery Man

Artificial Intelligence Course

Hyojung Kim & Nevine Gouda
UPPSALA UNIVERSITY

Table of Contents

Introduction

Overview on A* Search

A* Search algorithm is basically an informed search algorithm that tries to find the shortest path from point A to point B by avoiding expensive paths to the destination. This is done by exploring the states with a priority queue and with the help of a heuristic function that relates the state nodes to the goal state. In other words, at each move it only takes the step that ***appears*** to be closest/cheapest to the goal.

This is done using the following evaluation function:

$$f(n) = g(n) + h(n)$$

Where $f(n)$ is the estimated total cost of path through n to the goal.
And $g(n)$ is the actual cost so far to reach n.
While $h(n)$ is the heuristic function that estimates the cost from node n to the goal.
At each step in the algorithm, a frontier is created, that contains the list of nodes to be expanded/explored and each item in the frontier has its own cost that is calculated using the above evaluation function. From this frontier, we only explore/visit the node with the smallest cost (cheapest). We then remove it from the frontier and append it to the previously visited nodes list. Putting in mind that whenever we encounter a node that was previously visited but from a different path, its cost is updated with the path that has the smallest evaluation function (cost). This way it is guaranteed that it always finds the shortest path to the nodes, that of course with the right heuristic function.

Admissible heuristics & Optimality Conditions

A heuristic $h(n)$ can be considered admissible if for every node n, $h(n) \leq h^*(n)$ where $h^*(n)$ is the actual cost of reaching the goal from node n. And therefore, an admissible heuristic never overestimates the cost and can be considered an optimistic heuristic. For instance, a straight-line distance heuristic in streets/roads will never overestimate due to many variables such as street condition, traffic conditions or simply turns in the road. Thus, a straight-line distance is said to be an optimistic heuristic.

In addition, if we have 2 admissible heuristics, $h_1(n)$ & $h_2(n)$. And $h_2(n) \geq h_1(n)$ Then $h_2(n)$ dominates $h_1(n)$, meaning that $h_2(n)$ is closest to the truth, in other words, it is not too optimistic. Therefore using $h_2(n)$ in this case would be more efficient.

Algorithm Overview

The Delivery Man game overview

The ***DeliveryMan*** game is a grid type game. Where it consists of connected points/nodes with multiple packages located on random nodes across the grid and each of these packages has a certain destination. And the purpose of the game is for the car to deliver the packages to their destinations as fast/cheap as possible. Where the available moves to be done is for the car to go up, down, left or right within the border of the grid. Putting in mind that we can only carry one package at a time. Therefore, we classify this game as a graph search problem which can be solved using A* search algorithm. And due to the nature of the game; we can consider this to be multiple runs of the A* algorithm. Where we have more than one start node (packages locations) and goal node (packages destinations), and we need to reach them all. And as we move along the graph/grid (up, down, left or right), the roads conditions keep changing. Therefore, after every step we take, it is essential to re-calculate all the costs to the goal since the previous ones become outdated.

Thus, this game can be broken down to the following:

i. Given the car's current location and that we don't have any package loaded, we should decide which package to collect first, and only move one step towards it. And this is done by doing the following:
    1. Given the current car's location and the current road conditions, we calculate the shortest path for each package.
    2. The path is calculated using the following equation:

$$f(n) = g(n) + h(n)$$

- Where $g(n)$ is the actual cost from the car's position to the node (x, y). This can be obtained from the roads condition.
- While $h(n)$ is the **heuristic function** to reach the destination given the current node. Here we used the number of steps between the node n and the goal.

3. Choose the package with the shortest cost to go from the car's current position, to the package's position and to its destination. In other words, we don't just choose the closest package to the car, we also choose the package that is close to its destination.
4. Take only one step towards the package we choose (nextMove), then we re-calculate everything due to the change of road conditions. That is because due to new road conditions we might choose a different package to collect, or actually a different path to the goal.

ii. While given the car's current location, and that we have a package loaded that needs to be delivered, we should find the shortest path to deliver this package. This is done by a simple A* where the start node is the car's current location and the goal node is the package's destination. And the output is only the next move (up, down, left, right) towards the destination, and then re-calculate until the destination is reached.

Graph Search or Tree Search

Both of them are data structures that represent a series of connected data/nodes. Where the tree can be considered as a special type of a graph. Where trees can easily be stuck in infinite loops while graphs are considered more flexible as they can handle loops or even self-loops.

Thus, in our *DeliveryMan* game, we believe that it is more realistic to consider it as a graph search because of the following:

1- There is no parent-child relationship in the grid.
2- The grid contains direct loops.
3- Most importantly we need to store the previously visited/explored nodes.

Heuristics

Concerning the heuristic used in the algorithm, we first thought of the straight-line distance between node n and the goal. Where the straight-line distance is simply using the Euclidean distance equation to calculate the distance between two points (p1, p2).

$$dist(p_1, p_2) = \sqrt{(p_1(x) - p_2(x))^2 + (p_1(y) - p_2(y))^2}$$

For instance, in a square of length = 1, the distance between two points that are opposite to each other (connected through the diagonal) = $\sqrt{(1)^2 + (1)^2} = 1.414$

Despite this heuristic being admissible, it's not optimal. As there is a better, more dominant heuristic. A heuristic that simply counts the number of steps to reach the destination (the Manhattan distance). In the same example the cost will be $= 2.0$.

Thus, we created a matrix that calculates the number of steps between all the nodes in the grid/graph and the destination. Where in our game this will be used to calculate the Manhattan's distance between the car's position and the packages, or the car's position and the packages' destination (in case the load is present).

Tweaks and Improvements

We used a strategy when we calculated the overall cost in order to compare which package the car should go to next. Where for each package in the grid we needed to calculate 2 costs.

1- The cost from the car's current position, given the current traffic conditions.

   Where this cost is $f_1(n) = g_1(n) + h_1(n)$

2- The cost from the package to its destination.

   Where this cost is $f_2(n) = g_2(n) + h_2(n)$

Then we added both of them for it to be the total cost for a specific path.

$$f(n) = f_1(n) + f_2(n)$$

We did this to handle the case where we might choose the closest package but it's the destination is too far that the overall cost is actually the worst.