COMP 6721: Artificial Intelligence, Project 1 Report

Aria Adibi, 40139168

Abstract

In this project crime rate of a rectangular geographic coordinates of the city of Montréal is investigated. Then based on this investigation an optimal path must be provided for the user within a time limit of ten seconds. The purpose of the project is to familiarize students on the variety of state representation options available for a given problem and for them to analyze and compare different heuristic algorithms and functions.

Contents

List of Tables

1 Introductions and technical details

1.1 Introduction

Rectangular geographic coordinates of the city of Montréal and the committed crime locations represented as points within said coordinates (during a certain period of time) are given.

The map then should be divided into grid-like neighborhoods where the distance between two adjacent neighborhoods is given by `grid_size`, which is provided by the user of the program.

The first goal of the program is to show some statistics about the crimes in the neighborhoods and then, given a `percentile` by the user, identify and graphically show the *dangerous* neighborhoods.

Crossing an edge in this grid takes some *time*. According to the problem crossing an *unsafe* edge (within a dangerous neighborhood or on the boundary of two dangerous neighborhoods) is forbidden. The use of outer boundary edges are also forbidden. The time cost for other edges are as follows:

Edge crossing costs	
diagonal edge (within a safe neighborhood)	1.5(s)
boundary edge between two safe neighborhoods	1(s)
boundary edge between one safe and one unsafe neighborhoods	1.3(s)

Table 1: Costs of crossing a non-forbidden edge according to the problem

With this definition, the second goal of the program is to find a good state representation of the problem to solve for an optimal path (with a heuristic algorithm) between two points given by the user (the default points are top left to bottom right.) However, the program must find such an optimal path within a time limit of *ten seconds*. If an optimal path is found, it too is shown graphically to the user, if not an appropriate message is given.

The third goal of the project is to compare and analyze the pros and cons of different options available as heuristic algorithms and the heuristic functions. Moreover, it must give reasons for the choice of algorithm and heuristic function chosen by the program.

1.2 Technical details

In the provided `.zip` file you will find the following files:

- `README.txt`
  This `.txt` file provides the instructions needed to run the program.
- `main.py`
  The main (runnable) python module which is to be executed.
- `Technical Report`
  This file, which reports and give analysis of the project.
- `Expectation of Originality`
  A signed form for the purpose of originality of the work.

To obtain the data from `.shp` file, `geopanda` package is used.

Note. *Two heuristic functions, namely,* `naive_heuristic` *and* `moving_towards_heuristic` *are implemented. By default, the latter is used. For changing this, one has to comment one line of code and uncomment another line of code manually.*

2 Description, justification, and pros and cons of the heuristic algorithms

2.1 State representation

First, the program creates a matrix where each cell corresponds to a neighborhood. The number within the cell will be the number of crimes that occurred during the specified time period. Having this matrix, one could see that the pathfinding algorithm traverse a graph similar to the one obtained by considering the matrix as the representation of that graph.

For our algorithms to work, we somehow should create a state representation for them to work on. The above graph, along with the current position of the traveler, is one such state representation. Note that both algorithms do not utilize the entire graph in each step. Both use particular local information. Also, note that the above matrix is necessary to be created for us to have the crime information. Therefore, if we somehow

just give the algorithms the local information they need with minimal computational effort, then we represented the problem with a compelling state representation.

To this end, we use pairs of numbers as our graph vertices. With just simple index manipulation and looking locally at possible eight adjacent neighbors of a vertex, we can provide the algorithms their needed information. Accordingly, no graph is explicitly created and our representation, due to tight memory use and minimal computational effort, is a very compelling one.

2.2 Heuristic functions

Two heuristic functions are implemented. Both of them are *consistent* (and therefore also *admissible*.) Accordingly, they can be used with both *tree search* and *graph search* algorithms. Their description and analysis follows.

2.2.1 naive_heuristic

As it is apparent in the naming, this heuristic is very naive. It assumes no neighborhood is dangerous and gives the trivial answer in this case. It is easy to see that this heuristic is consistent.

2.2.2 moving_towards_heuristic

This heuristic forces the traveler to get at least one step closer (with respect of x or y axis) to the goal. If, however, the travel has no such option, the blocking dangerous neighborhood restriction is relaxed, and the traveler will be given a pass. In this case, the cost of diagonal travel is considered to be 1.5, and the cost of boundary edge is considered to be 1.3. This function h gives numbers to each vertex, in the following manner.

Initial values: $h(\texttt{goal}) = 0, \quad h(\texttt{others}) = \infty$

Value assignments: traverse the graph so that each time a vertex is reached, all the three possible neighbors of it have already been given a number by h. (It is easy to see such traverse exists.) Then assign

$$h((i,j)) = \min(\ h((i,j)),\ h(p,q) + \ \texttt{move cost from}\ (i,j)\ \texttt{to}\ (p,q)\ )$$

for all feasible neighbor (p, q).

The h function is admissible.

Proof.

Definition. *Let h^* to be exact optimal cost function for the given point to the goal point. If no path exists define $h^* = \infty$.*

Without losing the generality, assume the point in question is located at the top left of the goal.

If no optimal path exists, then since $h^* = \infty$ and $h \neq \infty$ (due to the relaxation), h is admissible. Therefore assume that there is an optimal path. Consider path P, suggested by function h, and an optimal path P^*. Note that, they both start at a single shared vertex and end at a single shared vertex.

Each action by the traveler in P has either right or down action in it, e.g. the diagonal action has both. Also, note that each action contains at most one left and at most one down action an no other type.

Now, consider the first point where these two paths deviate (exists, if not they were the same) and the first point after which they join again (exists because at the end the must end up at goal). From now on, we focus on this portion of these two paths. If it is proven that h is admissible within this portion (new start and new goal vertices), the proof is complete because the actual paths consist of finite multiple such portions.

Now focusing on the portion:

Think of diagonal actions in P as one right and one down actions with weight 1.3, but only weight wise. The new path cost is higher than before.

Assume P has r right actions, and P^* has r^* right actions (as defined above). Then the l^* (number of left actions in P^*) $= r^* - r$. Similarly, define for vertical actions. The horizontal *cost save* by P^* occurs when a right action in P with cost 1.3 is *replaced* with one in P^* with cost 1, save of 0.3. Therefore, the total horizontal savings will not exceed $0.3 \times r$. Similarly, $0.3 \times d$ for vertical moves.

However, the optimal path has $2 \times l^*$ additional horizontal moves, which costs at least $2 \times l^* \times 1$. Therefore, at best, the changes are:

$$-2l^* - 2u^* + 0.3r + 0.3d = -2r^* + 2.3r - 2d^* + 2.3d$$

We know $r^* \geq r$ and $d^* \geq d$. If $r^* > r$ and $d^* > d$ the above expression is negative, hence function h will be admissible. If not, if both equality hold then no left or up action is in the optimal path, therefore (by the relaxation of h) it is trivial that $h \leq h^*$ hence, h is admissible. If not this case either, then without the lose of generality assume $r^* > r$ and $d^* = d$.

By induction on the number of columns (based on $r^* > r$ and the horizontal distance from goal) and the fact that $h((i, j))$ is chosen based on min of guaranteed admissible neighbors, one can easily deduct the above equation cannot be positive because then h would have violated the min property of its neighbors. □

Similar argument but with a bit more nuance can be given to prove the consistency of h. However, since my search is tree search (admissibility is enough) I did not provide this argument.

Through some small time tests, the "better" heuristic function showed no time improvement. But my tests were very small and few. I assume it will do much better for randomly generated grids.

2.3 Heuristic Algorithms

Two heuristic algorithms, namely *Steepest Ascent Hill Climbing* and A^* is implemented. It is known to us that A^* finds the optimal path but Steepest Ascent Hill Climbing might stuck in local minimal or not be able to find any path. [1]

Due to small number of vertices and sparsity of the graph, A^* algorithm seems to be as fast as Hill Climbing algorithm, and only slightly more memory demanding (which is negligible in this kind of problems.) My very few tests seem to confirm this.

(Sorry, for the lack of time I cannot include the tests. If you would accept I would do it after the deadline.)

Both algorithms are tree search. For more about this decision please refer to Section 3.

3 Encountered difficulties

Originally, I wanted to implement the A^* as graph search algorithm. Which only add 5 lines of short code to the existing one. However, much to my surprise I found that `PriorityQueue` implementation in standard python library has no `decrease_value` function (as it should have, by definition). The reason (from what I understand) is related to handling `mutex lock` for multi threaded computations. The standard library also lacked any balanced (or similar) tree structures.

As I did not want to use third party data structure (in addition to popular packages like `numpy`), nor did I want to implement it myself, and judiciously guessing that the speed impact is not too significant, I opt for tree search algorithm instead.

References

[1] Stuart Russell and Peter Norvig, Artificial Intelligence: A Modern Approach 3rd Edition, Prentice Hall.