COMP 6721 Project Report

Yilin Li[1] and Yuhua Jiang[2]

[1] 40083064 yilinli0524@gmail.com

[2] 40083453 joeyyy1967@gmail.com

1. Introduction and technical details

1.1 Program background

The project is based on the intensity of crimes that have occurred in specific areas of downtown Montreal over the past few years.The objective is to find the optimal path between the start point and goal point when avoiding the high crime area. To achieve this purpose, we seperate the basic map into grids, use the statistical data to label the high criminal blocks. Then, based on the analyzed data, the user is able to input different map grid sizes and crime rate thresholds.The program will find the optimal path through two heuristic algorithms and output it on the map.

1.2 Development environment

The project is written in the Python3.0 programming language and is implemented in the Pycharm compiler.

1.3 Technical implementation

When extracting coordinate point data, we use the *shape file* library method. In drawing the grid map, the program references the *matplotlib* library, draws the yellow block by the *patches* in the library, and draws the final path through *Line2D*. In addition, in the algorithm design, we use two heuristic algorithms called the BestFirst and A* which are efficient algorithms for finding paths based on certain characteristics of points. There are two maps showing search results of two different algorithms at the same time, making it easy to compare their cost and efficiency. On the other hand, being able to set the limit time, the project references the *time* library and clocks the two algorithms separately to control the time of finding a path within ten seconds.

2. Data analysis and heuristic algorithm

2.1 Data storage structure

Two-dimensional array. The project uses two two-dimensional arrays to store grid information and point information separately.

The two-dimensional array of the grid stores the objects of the grid, and the length of each dimension is the number of columns and the number of rows of the grid. The attributes of the objects of the grids are the abscissa of the lower left corner of the grid, the ordinate, the number of the crime data of the grid and whether the grid is a yellow high crime area.

The two-dimensional array of points stores the objects of the points, and the size is the number of points in the horizontal and vertical directions. The object of each point contains the following attributes: the horizontal coordinate of the point, the vertical coordinate, the average crime rate of the rectangle from this point to the end point, the diagonal distance from this point to the end point, the heuristic function value of this point (h), the cost from the starting point to this point (g), the total cost of this point (f), the path cost of the point to the neighbor points, whether the point has been visited.

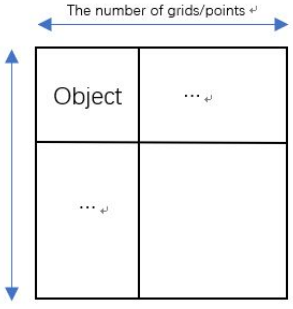

Fig. 1. Two-dimentional array

Dictionary. The dictionary data structure used by the program to store openList and closeList. The key is the object of the current point, and the value stores the object of its parent. As follow:

openList/closeList = {object of current node : object of parent node}

This data structure helps us to draw the final path in an easier way. More specifically, after the target node has been stored in the closeList dictionary, the program will start from this end point through the closeList, and the parent node stored in the value will be used to map the key value in the dictionary as the key. By analogy until it find the starting point.

2.2 Data processing and build map

Data Extraction. The coordinates are extracted from the .shp file using the shapefile library. The *shape()* method and the *shape.point()* method in the *shapefile* library are used to extract the horizontal and vertical coordinates in a loop, and then the index of the list is used to separate the horizontal coordinate and vertical coordinate. Then the program calculates the grid where the point is located according to this coordinate, and increases the number of crimes of this grid object by one at each time.

Points on grid boundary. We have designed certain rules to divide the points on the boundary into the grid. As shown in the following figure Fig.2, the basic grid contains the left and bottom boundaries of the grid, and the open circle indicates that the edge does not contain that point. The grids on the border of the map will contain points on their respective boundaries. By analogy, the grid in the lower left corner of the map contains only the points on the left and lower borders, while the grid on the top right corner of the map contains all the points on the four sides. In addition, for points that fall inside the grid, they are all attributed to the point in the lower left corner of the grid according to the rules we designed. We also use this rule to deal with the problem that the starting point and the ending point are not at the end of the grid after setting a certain gridsize such as 0.003.

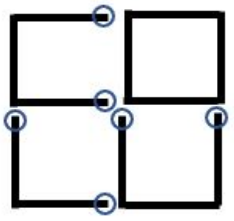

Fig. 2. Grid boundary point attribution

Points in the grid. There is a case where the size of the map does not match the gridsize, which means that the side length of the map cannot be divisible by the gridsize. For example, 0.003 or 0.007. We have appropriately increased the size of the map so that its size is exactly an integer multiple of the gridsize.

Draw the maps. The maps are drawn using the *matplotlib* library. The *pyplot.subplots()* method in the library we used sets up two side-by-side subgraphs to display the results of two algorithms. The color of the grid is then determined based on the threshold inputted by the user and the amount of crime data per grid. The method *addpatch()* with the *patches.Rectangle()* as a parameter is used to add yellow squares to the diagram.

2.3 Heuristic algorithm implementation

Heuristic function. Basic idea: The sum of absolute distance and criminal rate:

$$h = (a / (totalAll / (sizeX * sizeY))) + 0.7 * d \quad (1)$$

The a is the average crime rate of the grid in the rectangular area from the point to the target point, totalAll is the total number of crime data in map, sizeX and sizeY are the number of grids in the horizontal and vertical directions of map, respectively, and d is the diagonal distance from the point to the goal.

Our heuristic function can be divided into two parts : Absolute distance (the dialogo distance if the current point and distance are not on the same x or y coordination line) and partial criminal rate (The criminal rate in the rectangle area which is created by the current point and goal point as the vertex).

Reason to choose : We choose to use the combination of the two most important points connected with heuristic value of the search progress. According to the definition of heuristic value, it should be able to judge the current point by using a formula to determine how productive to reach the goal state by passing by it. We figured out that in our searching algorithms, there are two factors will defect the searching progress most: absolute distance and partial criminal rate. After several rounds of experiments, instead of simply adding them together,we decided to give 0.7 to the absolute distance as coefficient.
Absolute distance (d): Absolute distance is the straight line distance between the current point and the end point. We calculate it simply by using the diagonal formula:

*Diagonal distance = D * (abs(node.x-goal.x) + abs(node.y-goal.y)) + (D2 - 2*D) * min(abs(node.x-goal.x), abs(node.y-goal.y))* (2)

D represents the unit distance in the horizontal or vertical direction, and D2 represents the distance in the diagonal unit. In this project, D is 1 and D2 is the sqrt(2). This formula calculates the number of steps required to avoid the diagonal and then subtracts the number of steps saved by diagonal. The number of steps on the diagonal is min(dx, dy), and the cost per step is D2, which can save the cost of 2*D non-diagonal steps. When the current point and goal point are on the same x or y cordiration, it will be just the subtraction of two x or y cordiration.

We came to this idea by analyzing the weight of edges connecting every two points. More specifically, if the two points share the same x or y cordiration, the weight of the edge is just simply 1 (we assume the basic unit is one), at the same time, when the two points are connected by a dialogo line, the weight on the edge comes to the value of the square root 2. Furthermore, we choose to ignore scope if the "triangle" is not a regular triangle as we treat difference between the distances is tiny.
Partial criminal rate (a): We defined partial criminal rate as the proportion of the criminal rate in the rectangle area that using current point and goal point as vertex and the criminal rate of the whole area. To implement that, we static the total number of the criminal points which occured in the rectangle area, then use it to divide the number of the total criminal points in the entire map.

Algorithms structures

BestFirst pseudo code:

```
Put start node in  openList
while (openList is not empty)
        find the node with the minimum h in the openList
        Set currentNode as visited
        Add currentNode in closeList
        Remove currentNode from openList
        if (currentNode is located as the endpoint):
                if (process time less than 10 seconds):
                        findpath(currentNode, CloseList)
                else:
                        print("Time is up. The optimal path is not found.")
                break the loop
        else:
                for (the path cost of all neighbours of the currentNode):
                        if (the neighbour is accessed):
                                if (the neighbour node has not been visited yet):
                                        if (openList doesn't contain this node):
                                                Add this node to openList
if cannot find the path:
        print("Due to blocks. No path has been found!")
```

A pseudo code:*

```
Put start node in  openList
```

```
while (openList is not empty):
        find the node which has the minimum f in the openList
        Set currentNode as visited
        Add currentNode in closeList
        Remove currentNode from openList
        if (currentNode is located as the endpoint):
                if (process time less than 10 seconds):
                        findpath(currentNode, CloseList)
                else:
                        print("Time is up. The optimal path is not found.")
                break the loop
        else:
                for (the path cost of all neighbours of the currentNode):
                        if (the neighbour is accessed):
                                if (the neighbour node has not been visited yet):
                                        if (openList doesn't contain this node):
                                                Add this node to open list
                                        else: Update the g and f
    if cannot find the path:
        print("Due to blocks. No path has been found!")
```

3. Description and analysis of the results

3.1 Result examples

The whole project is based on finding the optimal path to reach the destination when avoiding the area that has a high criminal rate. To implement that, the size grid should be small enough to make the result more precise. At the same time, the threshold should not be too low to block the destiny grid since in that case, we are not able to access the destination grid. We set the grid size and threshold to be input form to ensure you can get at smallest as 0.001 as the grid size and 10 as the highest threshold. On the other hand, we recommend to use 0.002 and 75 for each of the session to show a more realistic path.

1st example: Grid size: 0.002; Threshold: 75

```
BestFirst path distance:  32.87005768508882
BestFirst path crime cost:  36.0
A* path distance:  30.627416997969533
A* path crime cost:  33.5
time cost for "best-first search" : 0.062 s
time cost for "A* search" : 0.094 s
```

Fig. 3. Path performance results (0.002, 75)

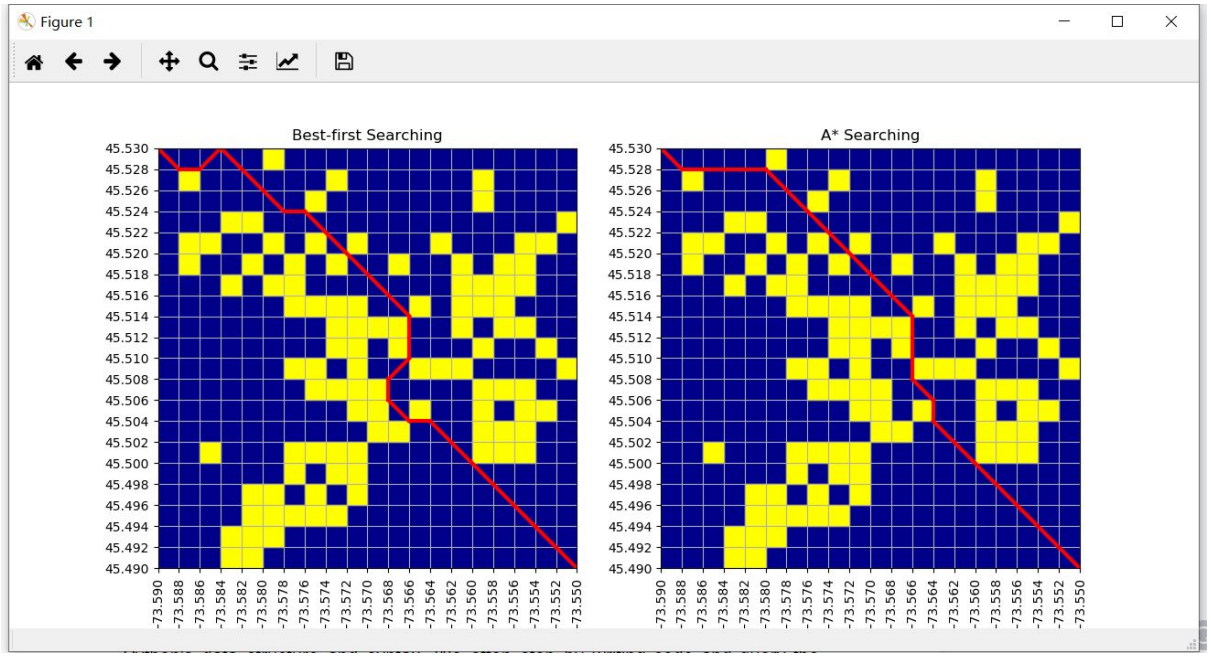

Fig. 4. Path figure results (0.002, 75)

2nd example: Grid size: 0.002; Threshold: 90

```
BestFirst path distance:  29.45584412271572
BestFirst path crime cost:  31.6
A* path distance:  29.45584412271572
A* path crime cost:  31.3
time cost for "best-first search" : 0.078 s
time cost for "A* search" : 0.094 s
```

Fig. 5. Path performance results (0.002, 90)

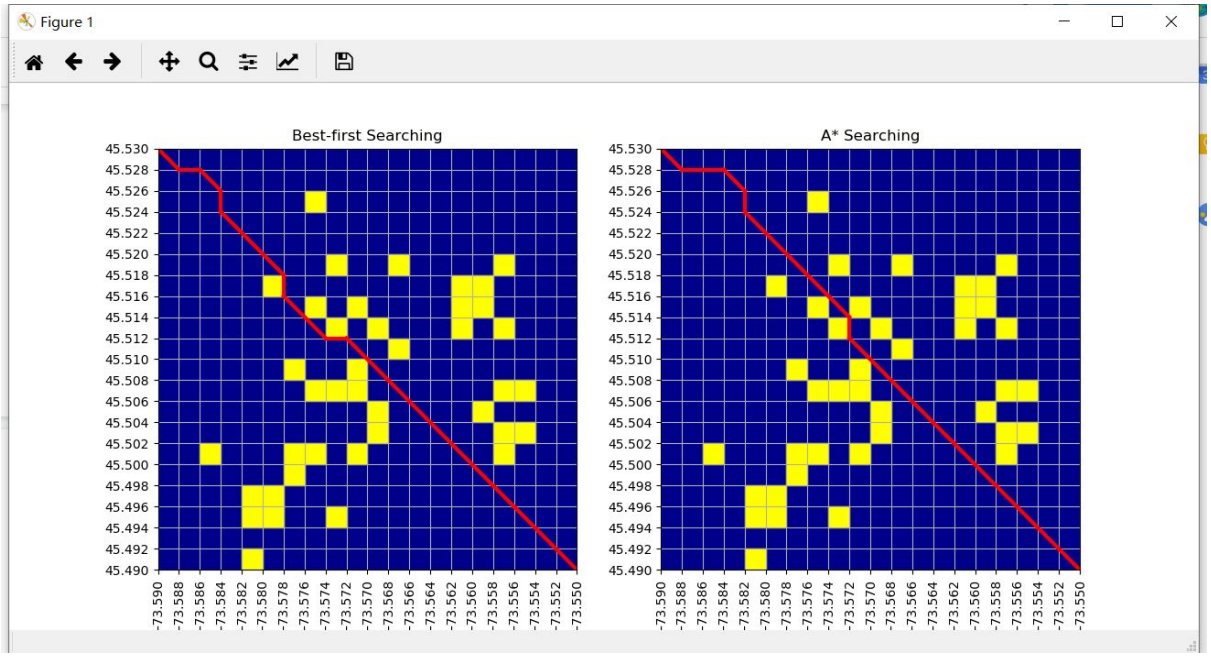

Fig. 6. Path figure results (0.002, 90)

3rd example: Grid size: 0.001; Threshold: 75

```
BestFirst path distance:  66.6690475583121
BestFirst path crime cost:  75.5
A* path distance:  63.254833995939
A* path crime cost:  69.6
time cost for "best-first search" : 0.529 s
time cost for "A* search" : 0.55 s
```

Fig. 7. Path performance results (0.001, 75)

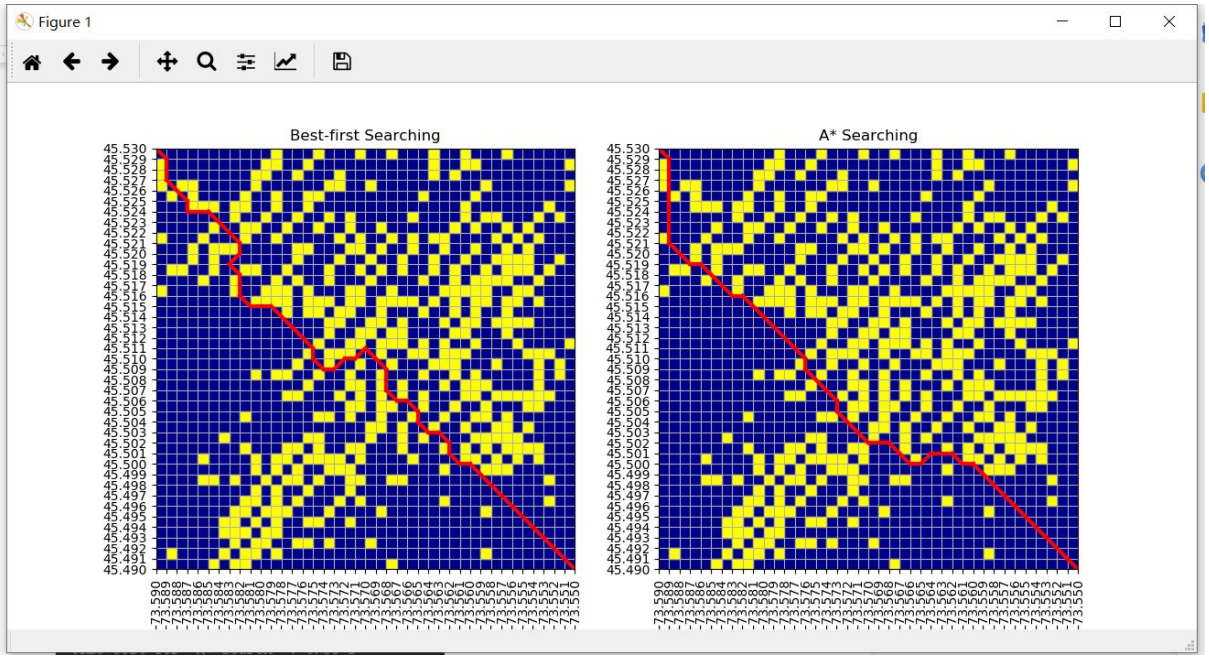

Fig. 8. Path figure results (0.001, 75)

4th example: Grid size: 0.001; Threshold: 90

```
BestFirst path distance:  60.6690475583121
BestFirst path crime cost:  66.8
A* path distance:  58.32590180780448
A* path crime cost:  62.4
time cost for "best-first search" : 0.484 s
time cost for "A* search" : 0.515 s
```

Fig. 9. Path performance results (0.001, 90)

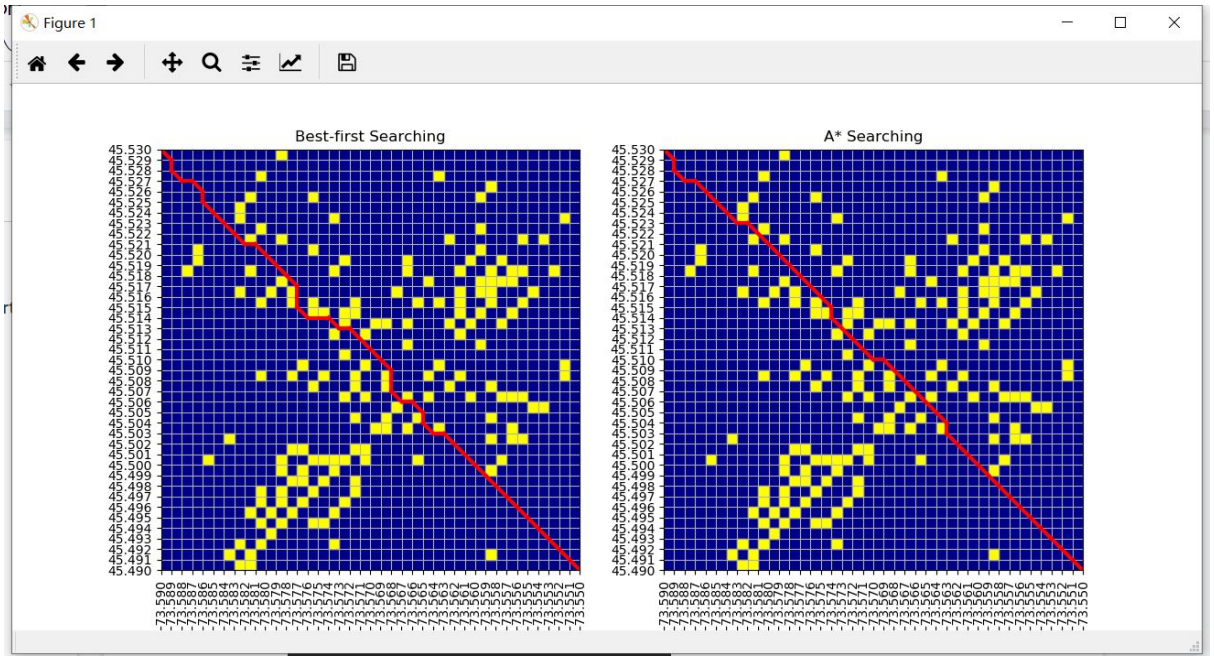

Fig. 10. Path figure results (0.001, 90)

5th example: Grid size: 0.003; Threshold: 75

```
BestFirst path distance:  28.142135623730958
BestFirst path crime cost:  30.5
A* path distance:  24.485281374238575
A* path crime cost:  27.1
time cost for "best-first search" : 0.042 s
time cost for "A* search" : 0.044 s
```

Fig. 11. Path performance results (0.003, 75)

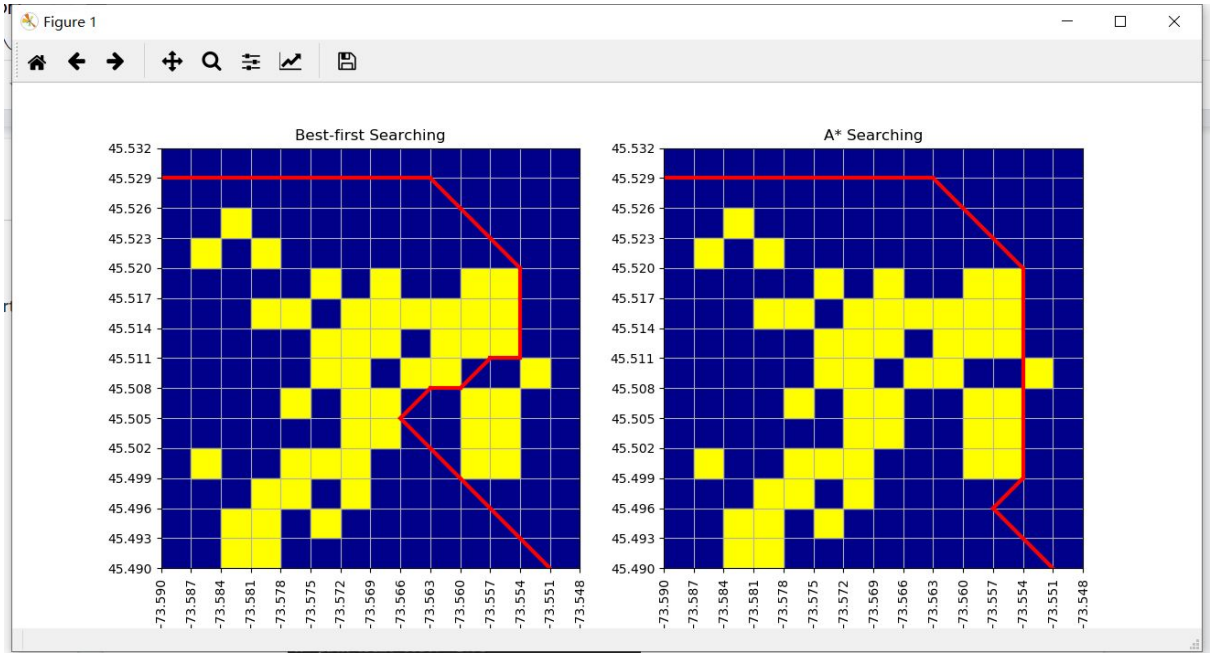

Fig. 12. Path figure results (0.003, 75)

3.2 Results comparison

The comparison of the results of the two heuristic algorithms is mainly carried out from three aspects: the path length, the crime rate of the path, and the time to find the path.

In terms of path length, after our test, the path length of the A* algorithm is generally less than or equal to the path of the BestFirst algorithm.

In addition, in the crime rate spent on the path, the path of A* is smaller than the path of BestFirst. Finally, in terms of time, the BestFirst algorithm takes less time. And the time of the two algorithms is much less than 10 seconds under different grid sizes and thresholds.

3.3 Algorithms comparison

Best-first search is a search algorithm which explores a graph by expanding the most promising node chosen according to a specified rule [1]. It is based only on the heuristic function h(n) attribute of the next node n, i.e an estimate cost of this point to the end point. The BestFirst algorithm is efficient and its time complexity is low. This makes it take less time to find the path in this program. But sometimes, it covers more distance than our consideration [2]. So it is slightly inferior to the A* algorithm in terms of other performance and the path found by this algorithm may not be best.

A* is a computer algorithm that is widely used in pathfinding and graph traversal, which is the process of finding a path between multiple points called "nodes", and it enjoys widespread use due to its performance and accuracy [3]. The formula for selecting the path of the A* algorithm is f(n)=g(n)+h(n), where n is the next node on the path, and g(n) is the cost of the path from the starting node to n, h (n) is a heuristic function that estimates the cost from n to the goal. Compared to the BestFirst algorithm, it considers the cost from the starting point to the point n. This makes its search results perform better, such as the path length and path crime rate measured in this project. But this extra part of the calculation also makes it take slightly more time in this project than the BestFirst algorithm. Moreover, here are three properties of the A* algorithm: termination and completeness, admissibility, optimal efficiency [3]. If there is a path in a finite graph with non-negative edge weights, A* will always find the path from the start point to the target. And the heuristic function used by A* is admissible because it never overestimates the cost of reaching the goal.

3.4 Strength and weaknesses of heuristics

For the best-first heuristic searching, it is not complete, may follow infinite path if heuristic rates each state on such a path as the best option. Furthermore, the worst case time complexity is still O(bm) where m is the maximum depth[4]. Apart from that, since must maintain a queue of all unexpanded states, space-complexity is also O(bm) [4]. However, looking at the bright side, a good heuristic will avoid this worst-case behavior for most problems.

For A* heuristic searching, it combines the features of uniform cost search (complete, optimal, inefficient) with best-first (incomplete, non-optimal, efficient). it sorts queue by estimated total cost of the completion of a path (f(n) = g(n) + h(n)) [4]. Further, if the heuristic function always underestimates the distance to the goal, it is said to be admissible, if h is admissible, then f(n) never overestimates the actual cost of the best solution through n.On the other hand, like the best- first searching, it also may act unstable when comes to time and space complexity. More specifically, time complexity will be O(bm) in the worst case since must maintain and sort complete queue of unexplored options[4]. However, with a good heuristic can find optimal solutions for many problems in reasonable time.

4. Difficulties

4.1 Unfamiliar with the programming language

Because both of the team members do not have a fountained background in python and we are not familiar with Python's data structure and syntax. We stuck from time to time when writing code and querying the relevant knowledge online, which greatly reduces the productivity of writing the project. Additionally, since we are not familiar with its libraries and related methods, most of the code is based on our original programming knowledge and does not use some simpler methods. This makes our code size larger and more redundant code.

It is worth mentioning that more than half of the bugs that occurred during our programming process are due to Python's computational problems. Because this project involves maps and coordinates, the accuracy of the calculations is important. However, in Python, the calculation results of the float type often have a slight deviation, so that all the results deviate from the original. We used a lot of *round()*

methods in places where calculations were involved, but occasionally we missed a few unexpected places. This kind of error is hard to notice.

4.2 Heuristic algorithms design

Designing heuristic functions is one of the difficulties of this project. In the beginning, our idea was to calculate the number of grids that exceeded the threshold on the square diagonal distance and the straight edge distance from the current point to the end point. The original heuristic function was designed by multiplying this number and normal path distance by different coefficients. However, during the programming process, we found that we need to consider the position of the current point relative to the end point, so we need to divide it into eight situations and discuss it one by one. So we gave up on this design and changed our idea. We redesigned the function and considered two factors that influence the path, which are crime rate and distance. The crime rate is expressed by the average crime rate of the rectangular area that is located from the point to the end point. The distance is calculated using the diagonal distance between two points. However, we found that the calculation results of these two factors are not in the same order of magnitude, so we divide the average crime rate of the rectangle by the number of grids in map, and then add this result to the diagonal distance. In the final test phrase, we adjusted the parameter values for the diagonal distance in the heuristic function. After testing and comparison, the parameter is set to 0.70.

5. Responsibilities and contributions

5.1 Division of work

Yilin Li (40083064): Data extraction; Heuristic algorithms design; Algorithms implementation; Testing and debug; Writing report.

Yuhua Jiang (40083453): Build maps; Heuristic algorithms design; Algorithms implementation; Refactor; Writing report.

5.2 Work mode

We process the whole project by using peer programming rule: all code is produced by two people programming on one task as a team. One programmer has control over the programming workstation and is thinking mostly about the coding in detail. The other programmer is more focused on the big picture, continually reviewing the code that is being produced, as well as researching solutions. During each task, we exchange the roll frequently to make everyone on the same page.

5.3 Work details

Data extraction: The act or process of retrieving data out of (usually unstructured or poorly structured) data sources for further data processing or data storage (data migration)[5]. The import into the intermediate extracting system is thus usually followed by data transformation and possibly the addition of metadata prior to export to another stage in the data workflow[5]. This process also includes cleaning and redirection of data.

Build maps: Use the library function to draw the map, including the axises, the distinction between different grid areas and the drawing of the path line.

Heuristic algorithms design: The core logic of the whole process of the heuristic searching. The prerequisite of the Algorithms implementation, including the designation of the formula for calculating the heuristic value as well as choosing the highly productive algorithms.

Algorithms implementation: According to the previous designation, write the pseudocode. Then transfer the logic pseudocode into python syntax.

Testing: Testing means verifying correct behavior. Testing can be done at all stages of module development: requirements analysis, interface design, algorithm design, implementation, and integration with other modules. In the following, attention will be directed at implementation testing. Implementation testing is not restricted to execution testing[6].

Debugging: Debugging is a cyclic activity involving execution testing and code correction. The testing that is done during debugging has a different aim than final module testing[6]. Final module testing aims to demonstrate correctness, whereas testing during debugging is primarily aimed at locating errors. This difference has a significant effect on the choice of testing strategies[6].

Refactor: Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior[7].
Writing report: Writing a report of a project means that in addition to stating what we did, we also need to describe the reason why we did it that way and what other options were available and the reason why they were abandoned.

References

1. Best-first search Wikipedia, https://en.wikipedia.org/wiki/Best-first_search, last accessed on 2019/08/29.

2. Best First Search: Concept, Algorithm, Implementation, Advantages, Disadvantages, https://www.brainkart.com/article/Best-First-Search--Concept,-Algorithm,-Implementation,-Advantages,-Disadvantages_8881/.

3. A* search algorithm Wikipedia, https://en.wikipedia.org/wiki/A*_search_algorithm, last accessed on 2019/10/03.

4. Heuristic search, http://www.cs.utexas.edu/users/mooney/cs343/slide-handouts/heuristic-search.4.pdf

5. Data extraction Wikipedia, https://en.wikipedia.org/wiki/Data_extraction, last accessed on 2019/08/30.

6. Testing and debugging, https://www.d.umn.edu/~gshute/softeng/testing.html

7. Refactoring, https://refactoring.com/