Research on the A Star Algorithm for Finding Shortest Path

Yumeng Yan*

Beijing-Dublin International College at BJUT, Beijing University of Technology, Beijing, 100124, China

* Corresponding Author Email: yym_lexie@emails.bjut.edu.cn

Abstract. With the rapid development of artificial intelligence technology, the problem of intelligent path finding is gradually important, which can optimize the robot walking route and the movement problem of NPCs in the game is improved. Among many shortest path algorithms, A-star algorithm to find the path is one very classic way. In this thesis, the authors mainly study the principle of A star algorithm and compare it with Dijkstra algorithm, and conclude that A-star algorithm is more effective and convenient. In this essay, writer used Raster Method to simulate the environment with obstacle, the distance between every two grids is calculated using Manhattan Distance Formula, which regulate the convenient way of moving is from top to bottom and move around. Through the outcome of the simulation of A star algorithm in Matlab, it shows that it is a better way to search the shortest path.

Keywords: Artificial intelligence; Robot walking route; Manhattan Distance Formula; Matlab.

1. Introduction

The shortest path is the core of route planning, which is one important component of navigation technology in modern society. Due to the traditional path search requiring every node to be judged, it leads to many problems of the huge search range, complex algorithm, long search time and low efficiency. Researchers developed a number of methods to find the shortest path faster, heuristic search methods, genetic algorithms, graph theory methods, and neural network methods… Heuristic search includes many subbranches: probabilistic path method, A* algorithm, Dijkstra algorithm, ant colony algorithm, and genetic algorithm.

Among these algorithms for searching the shortest path, Dijkstra algorithm is a typical method to find the shortest path, which needs to Travers many nodes and is less efficient to find the best solution at last, i.e., the shortest path. The A star algorithm is heuristic, which can decrease the number of nodes searched by heuristic functions to increase the searching efficiency. A-star algorithm can be used for the solution of many problems as well as optimization efficiently due to it is a heuristic search, which can improve the searching efficiency by decrease the nodes that need to be checked using heuristic function.

In this essay, the writer will mainly do some research on the A star algorithm that used for finding the shortest path. Renhao et al. (2015) argue that the heuristic search simulates the human brain's cognitive model of weighing some possibilities rather than all possibilities to make a judgment [1]. It will search through the evaluation of the nodes in the state space by evaluation equation, and expand them selectively, which will have low cost by selecting the appropriate heuristic function, which significantly reduces the search nodes and thus finds an optimal path quickly [2]. Heuristic search improves the search efficiency and solves the problems that traditional robot path planning methods such as large search range and complex algorithms.

In this essay, the A star path planning algorithm uses the raster method to model the environment, writer combines the A star algorithm, simulates the algorithm in Matlab, does two sets of experiments in a different simulation environment, and compare the principle of Heuristic search with Breadth First search and Depth First search algorithm, the outcome of the simulation shows that the shortest path planning for obstacle avoidance of robot faster can be achieved by A star algorithm.

2. Methods

2.1. Algorithm model/principle

A* Algorithm is the algorithm that can find the shortest path effectively, which can solve many search problems in the static road network [3].

About the question this paper is studying, the condition is that there is a wall on three sides, the starting point is at the top right of the wall, and the ending point is at the bottom right corner outside the wall. Figure 1 is an image of a wall and an obstacle.

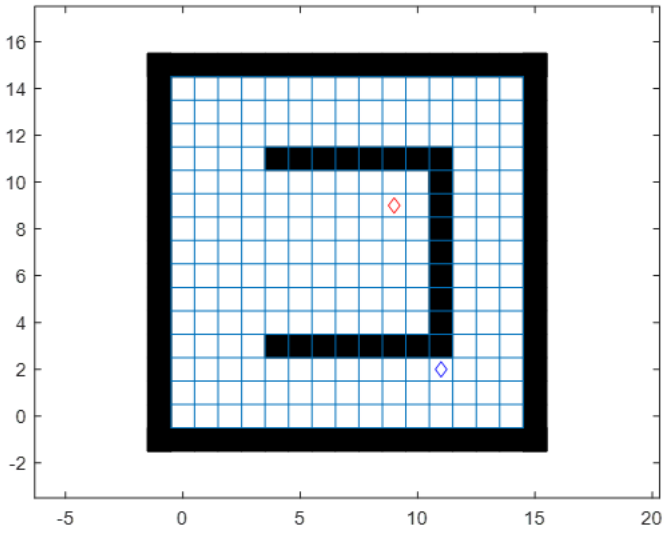

Fig 1. Images of walls and obstacles, red diamond is the starting point, the blue diamond is the endpoint.

There are a number of methods to find the shortest path: Breadth First, Dijkstra, and Best First Search…In this study, this writer sets the search speed to remain constant. The closer the predicted distance in the algorithm is to the practical value, that is, the distance of $h^*(n)$, the faster the target will be found [4]. Algorithm uses the following function to calculate the estimated distance staring from the current node to the ending grid, which decides the priority of each grid.

$$f(n) = g(n) + h(n) \quad (1)$$

The value of $f(n)$ estimated the distance from the current grid n to reach the target grid. The algorithm will select the grid which contains the lowest value of $f(n)$ to traverse in the next step.

$g(n)$ Is the distance of the current node from the initial point of searching [5].

$h(n)$, Is the forecast value between the current grid and the target grid? (It is the core of the A star algorithm; it is the heuristic function.)

A star algorithm chooses the node which contains the lowest value of $f(n)$ value from the openList, which is the node that be traversed next time during the process [6].

Heuristic Function: By comparing each nodes' heuristic function value, the speed and accuracy of the algorithm can be controlled. In some cases, the shortest path does not need to be found, but rather the fastest path needs to be found, which is also the flexibility of A* algorithm. The closer the estimated movement is to the practical value, the more similar between the final path and the actual shortest path [7].

Manhattan Distance: Only directions are allowed to move up, down, left, and right, then the heuristic function can use the Manhattan distance.

The set that represents the nodes which to be traversed next is an open list.

The set that represents the nodes which have been traversed is called a close list.

The first step is to divide the area to be searched into square grids. The grid will have a 2D array representation. Each item in the array represents a grid, and its status is walkable (obstacle free) and

unwalkable (with obstacle). The center point of the grid is used to represent the positions of the nodes [8].

Core of A star algorithm:

Starting the Search. The first step is to grid the searching area to simplify it. Searching process begins at the starting point withA^*, checks the adjacent squares of the current node, and then expand around nodes until the algorithm finds the target.

Once it expanded a node it will be added to an open list. At beginning, there is only the starting point item in the open list. More items will be added later with the expansion. The openList is to store the elements which will be checked [9].

Go over the nodes that is nearby to starting grid S and add the nodes which are walkable to the open list (in this case, the searching process will only consider the four directions of the current squares: above, below, left, right). Find the node that contains the lowest value of$f(n)$, and connect the starting point *S* to be the parent node of the current.

Theses parent nodes' value are very important while tracking the path from the target node to get to the starting grid [10]. The shortest path is founded by iterating through the open list and choosing the element sequentially.

$g(n)$ Is the distance from the starting node to the current grid?

$h(n)$ Is the estimated distance of moving from current grid to the destination?

The method of calculating (estimating)$h(n)$ are as follows: This paper uses the Manhattan method to calculate the value of squares that the current grid to reach the target by moving horizontally or vertically. While calculating, the obstacles in the map need to be ignored because it is in the process of estimating which is ideal.

Remove S (starting point) from openList. Then adds it to the closeList. The remaining four nodes are still in the open list. Every square in the close list needs no attention now.

If this element is in the closeList: unchanged.

If the element is out of the openList: add it to openList and calculate the sum at the current grid.

If the element is already in the openList: when the algorithm gets there using the currently generated path, check whether $f(n)$ and value are smaller. If so, update its sum value and its predecessor.

Next, a node which contains the lowest value of ($f(n)$) from the open list is needed to be selected. The grids which connect to the starting point is to loop the previous steps. Remove it from the openList and add it to the closeList [11].

Go over the nodes that are nearby to the current element and ignore the obstacles or the grids that are in closeList. If the nodes are not in the openList, add these elements to the openList. The result of A-star algorithm searching the shortest path is shown in Figure 2.

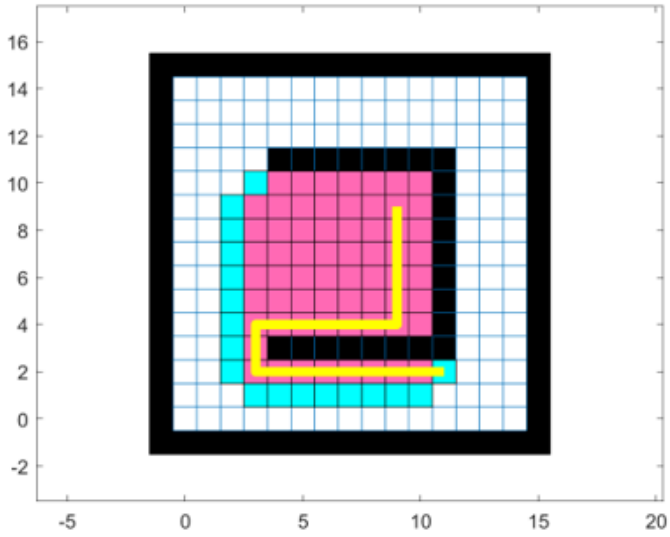

Fig 2. A star algorithm searches the shortest path.

2.2. Basic model

2.2.1. Raster method

This method essentially models the entire exploration environment in cells, represented by multiple, identical small squares. This method mainly utilizes the method of using small squares to represent the environment of obstacles and display them on the screen. In this experiment, the entire map is modeled using a two-dimensional array, 1 is used to represent obstacles, and 0 is used to represent normal meshes [12].

2.2.2. Manhattan distance formula

Manhattan distance is used to calculate the value of the absolute axis distances of two nodes in geometric metric space. The value of Manhattan distance in the 2-dimensional plane is the value between two points, calculating from the vertical axis and the distance on the horizontal axis and add these two distances. Figure 3 is an image of Manhattan distance calculation.

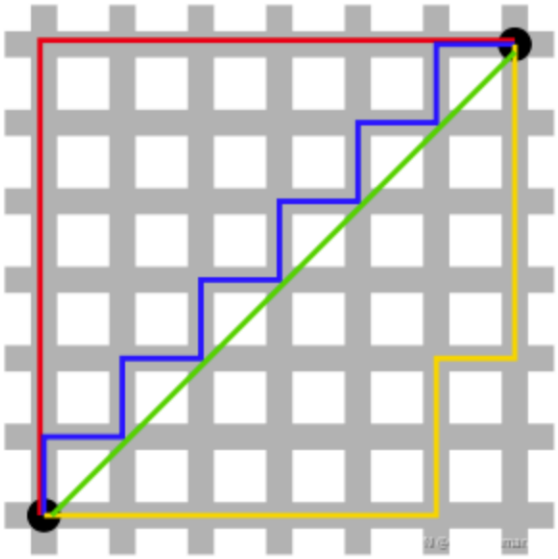

Fig 3. The image of calculating the Manhattan distance.

The green line is the Euclidean distance, which is used to calculate the straight-line distance between two points. The red line calculates the Manhattan distance by adding two values from the vertical axis and the horizontal axis.

$$d(a,b) = |a_x - b_x| + |a_y - b_y| \quad (2)$$

Manhattan Distance appears in early computer graphics, the screen of the machine was made up of pixels. It was sufficient to calculate addition, which greatly increased the speed of operations. Because there is no floating points operation, there is no error in performing multiple calculations.

3. Result and discussion

3.1. Result

3.1.1. Results of A star algorithm

A star algorithm is a very common way to find shortest path and travels graph, which has good performance and accuracy. This writer used a two-dimensional map that contains some obstacles that formed three walls to test this algorithm's performance. In this essay, the raster method is used to model the environment and implement the obstacle. This writer developed a map, which contains a three-sided wall that is 8 squares long and 9 squares wide. The starting point is between the three walls surrounding it, and the focus is beyond the walls. Set the starting point at (9, 9) location and the target point at (11, 2). Due to the infinite size of the grid points and the convenience of the study, as

well as the matching of Manhattan distances, in this study, the paths are explored in a way that they can only be moved up and down, not diagonally. Distance cost of grid movement is set to 10 in this paper.

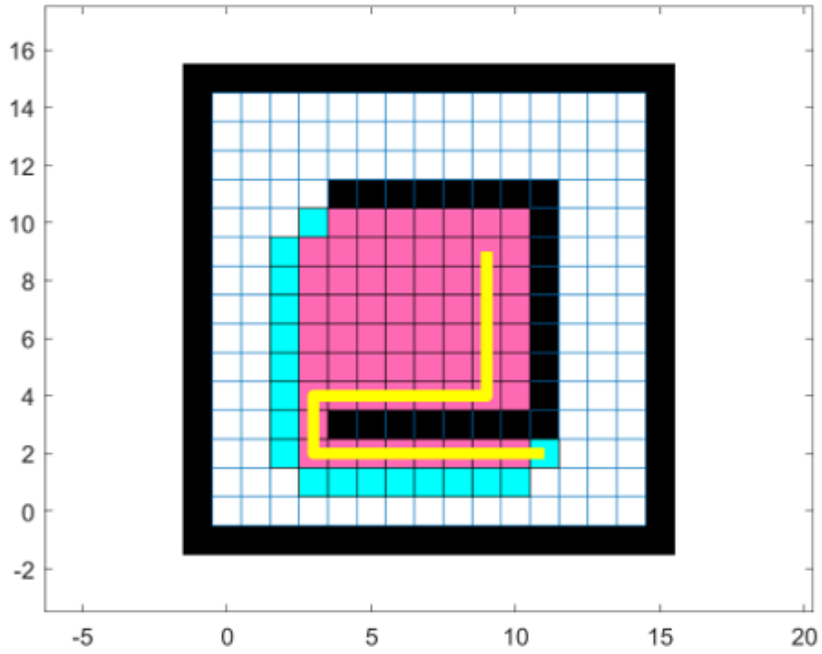

Fig 4. Outcome of A star algorithm.

In Figure 4, the grid that is printed to pink is the place that a star algorithm has already searched and expanded, the blue grids are representing the elements that had not been expanded and were added to the open list which stores the elements which will be searched in the next step. The yellow path is the shortest path found by a star algorithm starting from the target point.

a. The "path" list (yellow grids) stores the shortest paths that are determined by algorithm *A**.

b. The "openList" (blue grids) is to record all the squares considered to find the shortest path.

In this list, it stores six elements, the first two are the node's coordinates, the third is the function *F* represents the Sort all rows in the list by the third column F value in the OpenList, the fourth is *G*, which represents the distance of this node from the start point, last two is the coordinates of its parents' node.

c. The "closeList" (pink grids) is to record a square that is recorded and will not be considered again.

The first step is to put the beginning into the openList and expand it.

Then do the following steps until reach the target or have no path.

Use the "isempty (openList (: 1))" method to check if a path is reachable.

Call the "isOpen" function to check whether the openList contains the target, if the target is in openList, which means the path has found the target and can reach this point. Then add the target to the closeList and remove it from the openList.

1. Use "sort" to rank all rows of the openList in the list from smallest to largest by the third column representing the Sort all elements in the list by the third column (F's value) in the OpenList.

2. Move the first element in the open table to the close list, set it to the current element and delete it from the open list.

3. Check the surrounding points. The elements in "node" are [coordinates of surrounding points of the current node, F, G, the coordinates of parent node].

Node= [current (1, 1) +next (in, 1), current (1, 2) + next (in, 2), 0, 0, 0, 0];

For the surrounding points, there are 4 conditions:

a. If it is a barrier or obstacle, then ignore it.

b. If it is in the close List, ignore it because it has been searched.

c. If it is not in the open List, add it in the open List, then set the current node as its parent.

d. In the other condition (it's in the open List), when the value of the third column is smaller than the indexed element, use this node to represent the original indexed node.

Use the "Find Path" function to calculate the final path by gradually finding the parent node of the current element, finally reaching the starting point and outputting the shortest path in reverse.

3.1.2. Summary of the A* method

First step: Add the starting node into the open list.

Next step: Do the following process until the process does not fit the condition:

Go through the open list to find the lowest value of node and set it as the current node to be operated.

Put this node to the close list.

For every 4 adjacent squares of the current grid, search them sequentially. When the current node in the close List, do not do any operation. If the current node is out of the open List, then add it into the open List, besides, set the current grid as the father of this adjacent node.

Repeat these steps to find the path.

Stop and add the destination point to the close list. At this time, the path has been found, or failed to find the destination.

Save the path:

Searching begins at the target grid, each grid finds its parent node and move to it until get to the starting grid, and find the shortest path at last.

3.2. Discussion

3.2.1. Comparation with two basic search Strategies

Depth first search method (DFS).

The searching process traverses along the depth of the tree, and search as deep as they can into every possible branch path. When all the edges of the current node have been explored at the time of searching, the process of searching will return to the starting grid. It is one of the blind search because the process is repeated until all nodes have been visited, and the worst-case algorithm has a time complexity of O (! n).

Breadth first search method (BFS).

It is a blind search method which expands all nodes sequentially in the graph to find results. Unlike depth search, depth search searches one path at a time and keeps searching until it fails to go; while BFS searches all paths at the same time, which is equivalent to searching one layer at a time, just like the expansion of a wave. This search method is similar to the hierarchical traversal of the tree, so the width search is generally used in a queue storage structure.

Compared with width search and depth search, the A-star algorithm solves the disadvantages of the above two. The advantage of deep search is that it is fast in time, but it does not ensure that the shortest path is always found; while width search does find the optimal solution, it is not efficient in time and space because it searches every layer and must expand each point. The A-star algorithm is obtained by designing an evaluation function that has two evaluation values summed. One is the cost in the practical from the starting grid to get to the optimal solution obtained after already searching, and one is the predicted the distance from current location to target location (generally the predicted value is smaller than the actual search value), and by evaluating this function the method can come up with the next point that should be taken. Because the prediction of $h(n)$ cannot be completely accurate, the expected value may deviate from the real dissipation value: if the value of $h(n)$ is as same as the actual distance $h^*(n)$, shortest path which is found by A star algorithm is the optimal solution, and basically no need to expand other points; if the value of h(n) is smaller than the real distance, the efficiency will decrease, if $h(n) = 0$, which means that the value of $f(n)$ is equal to $g(n)$, which is BFS; if $h(n)$ is larger than the real value, the time efficiency will increase, but this will make the process of searching close to the depth-first search and may not obtain the optimal solution.

3.2.2. Limitations of this thesis study:

As the software performs simulation, it cannot do a test with great accuracy and cannot ensure that the results obtained from the experiment are still consistent output in real life. The greater the accuracy, the longer the exploration time. Figure 5 is a map enlarged.

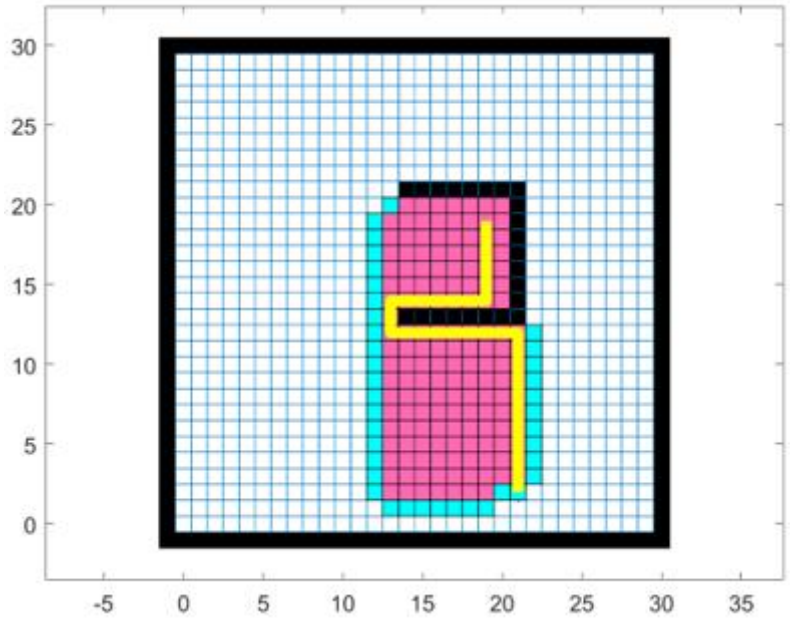

Fig 5. The figure that enlarges the size of the map.

In this study, there is no discussion on the obstacle boundary, the obstacle setting form is relatively single, the search time in the algorithm into the maze of obstacles rises greatly, and no better solution is proposed.

4. Conclusion

This essay uses the Raster Method to implement the environment and set the obstacle and starting point and target automatically. The simulation of the A-star algorithm was implemented and tested using Matlab. The algorithm uses the evaluation function to predict the nodes to be searched in the future and selects the points to be prioritized for walking, which works well in searching the optimized shortest path.

References

[1] Xiong, R., & LIU, Y. (2015). Improvement and parallelization of A* algorithm. Journal of Computer Applications, 35(7), 1843.

[2] Jing, X., & Yang, X. (2018). Application and Improvement of Heuristic Function in A* Algorithm. In 2018 37th Chinese Control Conference (CCC) IEEE.

[3] Faisal, M., & Zamzami, E. M. (2020). Comparative analysis of inter-centroid K-Means performance using euclidean distance, canberra distance and manhattan distance. In Journal of Physics: Conference Series, 1566(1), 012112.

[4] Candra, A. (2021). Application of A-Star Algorithm on Pathfinding Game. Journal of Physics: Conference Series, 1898(78), 012047.

[5] Duchoň, F., Babinec, A., Kajan, M., Beňo, P., Florek, M., Fico, T., & Jurišica, L. (2014). Path planning with modified a star algorithm for a mobile robot. Procedia Engineering, 96, 59-69.

[6] Yao, J., Lin, C., Xie, X., Wang, A. J., & Hung, C. C. (2010). Path planning for virtual human motion using improved A* star algorithm. In 2010 Seventh international conference on information technology: new generations IEEE.

[7] Ghaffari, A. (2014). An energy efficient routing protocol for wireless sensor networks using A-star algorithm. Journal of applied research and technology, 12(4), 815-822.

[8] Bell, M. G. (2009). Hyperstar: A multi-path Astar algorithm for risk averse vehicle navigation. Transportation Research Part B: Methodological, 43(1), 97-107.

[9] Cheng, L., Liu, C., & Yan, B. (2014). Improved hierarchical A-star algorithm for optimal parking path planning of the large parking lot. In 2014 IEEE International Conference on Information and Automation (ICIA) IEEE.

[10] Rousseau, G. L. A., Bostel, J., & Mazari, B. (2005). Star recognition algorithm for APS star tracker: Oriented triangles. IEEE Aerospace and Electronic Systems Magazine, 20(2), 27-31.

[11] Attoyibi, M. M., Fikrisa, F. E., & Handayani, A. N. (2019). The Implementation of A Star Algorithm (A*) In the Game Education about Numbers Introduction. In 2nd International Conference on Vocational Education and Training (ICOVET 2018) Atlantis Press.

[12] Kim, H. Y., & Junkins, J. L. (2002). Self-organizing guide star selection algorithm for star trackers: thinning method. In Proceedings, IEEE Aerospace Conference IEEE.