Proceedings of the 2nd International Conference on Software Engineering and Machine Learning
DOI: 10.54254/2755-2721/67/20240463

Optimization of the mobile robot's path searching based on the A-star algorithm

Yunbo Zhang

Computer Science and Technology, Dalian Maritime University, Dalian, China

zyb03123@dlmu.edu.cn

Abstract. Notable has been the rapid development of artificial intelligence in recent years, with mobile robots being widely applied in various fields. Greatly facilitating people's daily lives, research on robots has received widespread attention. The obstacle avoidance capability of mobile robots is an important indicator of their intelligence. This article utilizes the A-star algorithm. It optimizes the robot's movement path. The A-star algorithm is characterized by balancing the actual cost from the starting point to the current node with the estimated cost from the endpoint to the current node. This algorithm incorporates elements of both Dijkstra's algorithm and greedy search. By being guided by a heuristic function, A-star efficiently approaches the target point, ensuring optimal path planning. This paper focuses on optimizing the A-star algorithm, enhancing both node expansion during path exploration and search time for paths as the robot navigates through a maze.

Keywords: A-star Algorithm, Heuristic Function, Robot Path Planning, Maze Exploration, Weighted A-star Algorithm.

1. Introduction

When employing the Dijkstra algorithm to govern the movement of robots within a maze, the author observed a notable expansion of nodes, nearly encompassing all nodes inside the maze, significantly impacting the algorithm's efficiency. In contrast, the A-star algorithm effectively diminishes the number of expanded nodes by incorporating a heuristic function [1]. Consequently, this paper introduces the A-star algorithm to optimize the robot's movement path [2]. In comparison to the Dijkstra algorithm, the A-star algorithm amalgamates elements of Dijkstra's algorithm with greedy search and judiciously selects nodes for expansion by integrating a heuristic function, thereby substantially reducing the search space [3-4]. This advantage empowers the A-star algorithm to swiftly discover optimal solutions during the path-search process, thereby enhancing its efficiency. This is particularly pivotal in resource-constrained environments such as mobile robotics, contributing to the algorithm's feasibility and implementation. The A-star algorithm not only extensively applies to robot path planning but also finds utility in diverse fields, including game development, network routing, and logistics planning [5-6]. Consequently, introducing the A-star algorithm not only assists in optimizing robot movement paths but also broadens its application across various domains.

This paper focuses on analyzing the expanded number of nodes and the time required for path exploration as robots traverse a maze. Through iterations, the optimal scenario for both the number of expanded nodes and the time taken for path exploration is determined.

DOI: 10.54254/2755-2721/67/20240463

2. Literature Review

Scientists have been consistently working to develop more efficient heuristic functions, aiming to improve the A-star algorithm's performance across diverse application scenarios. [7]. This may involve considering factors like dynamic obstacles, terrain changes, robot energy consumption, among others, to more accurately evaluate node costs. For multi-target robot movement, several improved versions of the A-star algorithm have been proposed to simultaneously consider multiple objectives within the same path, better catering to practical needs. With the diversification of robot application scenarios, the demand for real-time path planning is increasing. Some studies focus on enhancing the real-time performance of the A-star algorithm to adapt to environments requiring immediate decision-making, such as autonomous vehicles, drones, etc. [8]. Robots often encounter dynamic environments in the real world, including changes in pedestrians, vehicles, and other elements. Relevant research focuses on introducing real-time perception and adaptability to dynamic obstacles within the A-star algorithm. This ensures the real-time and safety aspects of path planning [9].

3. A-star Algorithm

3.1. Algorithm Principles

The A-star algorithm is a heuristic search algorithm used for determining the shortest or most optimal route from an initial point to a designated destination. It is widely employed in pathfinding and graph search problems, encompassing domains such as computer games, robot path planning, and map navigation. The A-star algorithm is built upon the concepts of Dijkstra's algorithm and heuristic search, combining the advantages of breadth-first search and greedy algorithms.

The A-star algorithm typically executes searches within a graph or network, where nodes represent states along potential paths, and edges signify transitions between one state to another. Typically, every node is linked with a heuristic estimate, representing the distance or cost from that node to the target node. The algorithm's goal is to calculate the shortest path from the starting point to the end point.

Each node holds an f-value, denoting an estimation of the total cost from the starting point, passing through that node, to the target node. Its general formula is expressed as equation (1):

$$F(n) = g(n) + h(n) \tag{1}$$

Where:

$f(n)$ represents the comprehensive priority of the node n. When selecting the next node to traverse, the algorithm always picks the node with the highest comprehensive priority (lowest value).

$g(n)$ stands for the cost of the path from the starting point to the node n.

$h(n)$ signifies the heuristic function estimating the cost from node n to the goal. This function embodies the core of the A-star algorithm.

During computation, the A-star algorithm prioritizes selecting the node with the smallest f(n) value from the priority queue for the next traversal. It employs an open list to maintain nodes for expansion, initially containing the starting point. During each iteration, the algorithm picks the node from the open list that possesses the lowest f-value for expansion. Nodes that have been expanded are added to the closed list to prevent redundant expansions. The A-star algorithm considers both actual costs and heuristic estimates when selecting the next node to expand, aiming to reach the target state as quickly as possible. In this way, it tends to prioritize exploring paths that are believed to be closer to the target, thereby reducing the search space. In contrast, algorithms like Dijkstra's explore all possible paths equally without considering the distance to the target. Therefore, the A-star algorithm can intelligently guide the search process, leading to faster solutions within the same search space.

3.2. Algorithm Procedure

The A-star algorithm iterates through the following steps until it either discovers the destination or the open list is depleted:

(1) Select the node with the minimum F value from the open list.

DOI: 10.54254/2755-2721/67/20240463

(2) The search concludes with the discovery of the shortest path, if the selected node is the destination. Otherwise, move the selected node from the open list to the closed list and expand it, considering its neighboring nodes.

(3) Calculate the G and H values for each neighboring node, and then include them in the open list.

(4) Once the destination is reached, the shortest path can be obtained by backtracking through each node's parent nodes, traversing from the destination back to the starting point.

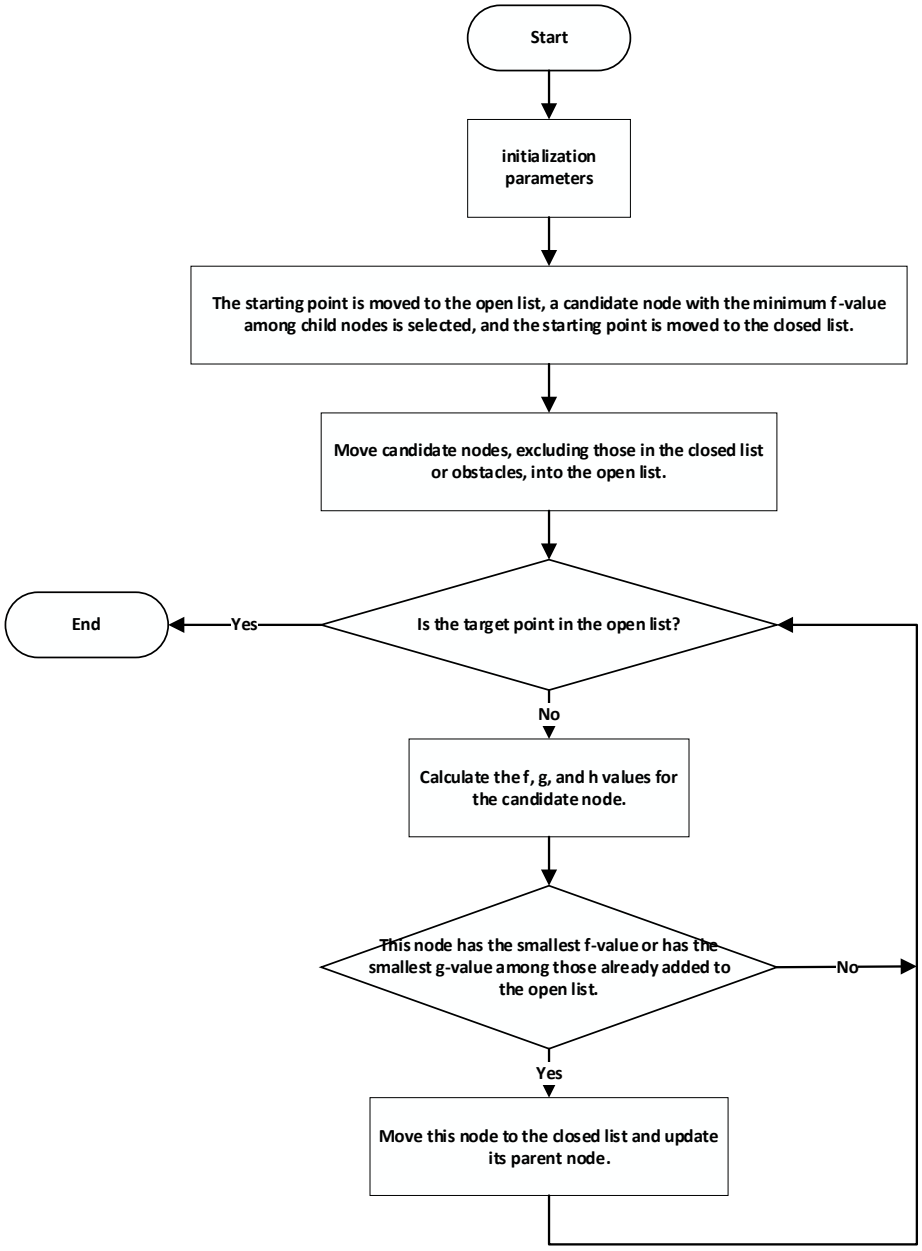

Figure 1. A-star Algorithm Flowchart

3.3. Heuristic Function Selection

Manhattan distance is a metric used to calculate the distance between two points, particularly useful in path planning and searching on regular grids [10].

DOI: 10.54254/2755-2721/67/20240463

Compared to other distance measurement methods, calculating the Manhattan distance is simpler. This greatly improves computational efficiency for large-scale search problems. Additionally, the Manhattan distance performs well in many applications. Therefore, this article selects the Manhattan distance as the heuristic function.

The Manhattan distance algorithm operates by calculating the sum of the horizontal and vertical distances between two points to determine the distance between them. The Manhattan distance formula for two points A(x1, y1) and B(x2, y2) is formulated as shown in equation (2):

$$\text{Manhattan Distance} = |x_1 - x_2| + |y_1 - y_2| \tag{2}$$

Where $|x_1 - x_2|$ represents the distance in the horizontal direction, and $|y_1 - y_2|$ represents the distance in the vertical direction. This formula calculates the total length of the shortest path along the grid edges from point A to point B.

3.4. Improvements of the A-star Algorithm

The performance of the algorithm is improved by introducing a weighting factor (denoted as "w"). The fundamental principle of the weighted A-star algorithm involves adding a weighting factor to the heuristic function. This adjustment allows for the tuning of the search algorithm's behavior. This allows the A-star algorithm to flexibly balance the search speed and optimality in different scenarios. In order to adjust the heuristic function, the author uses equation (3):

$$F(x) = g(x) + w * h(x) \tag{3}$$

The weighted factor 'w' regulates the relative importance of the heuristic function. Different values of 'w' correspond to varying algorithm efficiencies. When 'w' equals 1, it represents the traditional A-star algorithm, while 'w' being 0 signifies Dijkstra's algorithm.

4. Simulation Experiments and Analysis

4.1. Traditional A-star Algorithm

The simulation experiment results, depicted in Figure 2, showcase the light yellow nodes as the number of nodes expanded by the traditional A-star algorithm, while the deep red lines represent the ultimately chosen path. It is evident that, compared to Dijkstra's algorithm, the traditional A-star algorithm demonstrates significant optimization. However, there are still a considerable number of generated nodes that are deemed unnecessary.

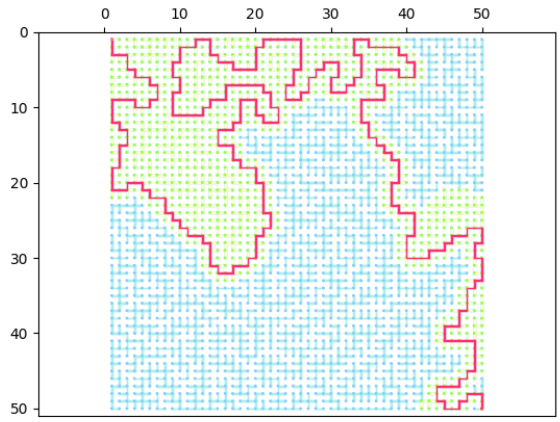

Figure 2. The simulation experiment results(w=0.250)

DOI: 10.54254/2755-2721/67/20240463

4.2. Weighted A-star Algorithm

From the observation in Figure 3, it's evident that after applying weighting to the heuristic function, the quantity of expanded nodes significantly reduces. Alongside the left edge of the maze, there's a noticeable decrease in the count of expanded nodes compared to the previous experiment. On the right side of the maze, there's virtually no expansion of unnecessary nodes, significantly enhancing the algorithm's efficiency."

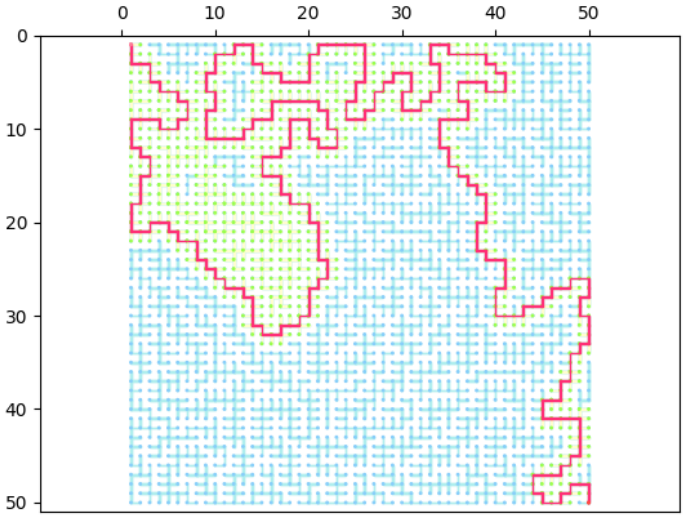

Figure 3. The simulation experiment results (w=0.875)

4.3. Experimental Analysis

After normalizing 'w,' various values were selected for simulation experiments to evaluate the optimal 'w' by computing the quantity of expanded nodes and the time taken for path searching.

Table 1. Relationship between 'w' and the Quantity of Expanded Nodes and Path Search Time

w	Number of Expanded Nodes	Path Search Time
0	1135	735.9
0.250	1020	666.3
0.475	959	547
0.675	879	550.8
0.875	843	470.6

When using Dijkstra's algorithm, the expanded nodes essentially cover the entire maze. With the traditional A-star algorithm, there's a significant decrease in the expansion of unnecessary nodes, resulting in a substantial decrease in search time. When the weight value is 0.475, the number of expanded nodes decreases by 6.0% compared to the traditional A-star algorithm, and there's a noticeable reduction in path search time. At w=0.875, there's a reduction of 17.4% in the count of expanded nodes compared to the traditional A-star algorithm, leading to a 29.4% decrease in path search time(Table 1).

5. Conclusion

After conducting repeated experiments, determining the optimal path appeared relatively straightforward. As 'w' varied, both the number of explored nodes and the time taken for path search demonstrated corresponding changes. Following normalization, at 'w=0.875', the robot expanded the

DOI: 10.54254/2755-2721/67/20240463

fewest nodes while achieving the shortest path search time within the maze. However, several limitations persisted in this experiment. Primarily, the maze design was relatively simple, and its scale was limited. Future experiments could involve more intricate simulation environments, such as introducing moving obstacles to impede the robot's movement. Additionally, including further evaluation criteria, like individually controlling the weights of g(n) and h(n) to compare the number of expanded nodes, could enhance the experiment's depth.

References

[1] Ju, C., Luo, Q., & Yan, X. (2020, October). Path planning using an improved a-star algorithm. In 2020 11th International Conference on Prognostics and System Health Management (PHM-2020 Jinan) (pp. 23-26). IEEE.

[2] Garcia, E., Jimenez, M. A., De Santos, P. G., & Armada, M. (2007). The evolution of robotics research. IEEE Robotics & Automation Magazine, 14(1), 90-103.

[3] Chen, Y. H., & Wu, C. M. (2020, September). An improved algorithm for searching maze based on depth-first search. In 2020 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-Taiwan) (pp. 1-2). IEEE.

[4] Lina, T. N., & Rumetna, M. S. (2021). Comparison Analysis of Breadth First Search and Depth Limited Search Algorithms in Sudoku Game. Bulletin of Computer Science and Electrical Engineering, 2(2), 74-83.

[5] Tsoi, N., Hussein, M., Espinoza, J., Ruiz, X., & Vázquez, M. (2020, November). Sean: Social environment for autonomous navigation. In Proceedings of the 8th international conference on human-agent interaction (pp. 281-283).

[6] Gunawan, R. D., Napianto, R., Borman, R. I., & Hanifah, I. (2019). Implementation Of Dijkstra's Algorithm In Determining The Shortest Path (Case Study: Specialist Doctor Search In Bandar Lampung). Int. J. Inf. Syst. Comput. Sci, 3(3), 98-106.

[7] Zhao, Z., Zhou, M., & Liu, S. (2021). Iterated greedy algorithms for flow-shop scheduling problems: A tutorial. IEEE Transactions on Automation Science and Engineering, 19(3), 1941-1959.

[8] Mammeri, Z. (2019). Reinforcement learning based routing in networks: Review and classification of approaches. Ieee Access, 7, 55916-55950.

[9] Golpîra, H., Khan, S. A. R., & Safaeipour, S. (2021). A review of logistics internet-of-things:

[10] Current trends and scope for future research. Journal of Industrial Information Integration, 22, 100194.

[11] Yiu, Y. F., Du, J., & Mahapatra, R. (2018, September). Evolutionary Heuristic A Search: Heuristic Function Optimization via Genetic Algorithm. In 2018 IEEE First International Conference on Artificial Intelligence and Knowledge Engineering (AIKE) (pp. 25-32). IEEE.

[12] Parekh, D., Poddar, N., Rajpurkar, A., Chahal, M., Kumar, N., Joshi, G. P., & Cho, W. (2022). A review on autonomous vehicles: Progress, methods and challenges. Electronics, 11(14), 2162.

[13] Aggarwal, S., & Kumar, N. (2020). Path planning techniques for unmanned aerial vehicles: A review, solutions, and challenges. Computer Communications, 149, 270-299.

[14] Shen, Y., Zhang, F., Liu, D., Pu, W., & Zhang, Q. (2022). Manhattan-distance IOU loss for fast and accurate bounding box regression and object detection. Neurocomputing, 500, 99-114