Mathematical Biosciences and Engineering

MBE, 21(2): 2137–2162.
DOI: 10.3934/mbe.2024094
Received: 25 October 2023
Revised: 03 December 2023
Accepted: 22 December 2023
Published: 10 January 2024

http://www.aimspress.com/journal/MBE

Research article

Workshop AGV path planning based on improved A* algorithm

Na Liu, Chiyue Ma*, Zihang Hu, Pengfei Guo, Yun Ge and Min Tian

College of Mechanical and Electrical Engineering, Shihezi University, Shihezi 832003, China

* **Correspondence:** Email: machiyue@stu.shzu.edu.cn; Tel: +8615997563200.

Abstract: This article proposes an improved A* algorithm aimed at improving the logistics path quality of automated guided vehicles (AGVs) in digital production workshops, solving the problems of excessive path turns and long transportation time. The traditional A* algorithm is improved internally and externally. In the internal improvement process, we propose an improved node search method within the A* algorithm to avoid generating invalid paths; offer a heuristic function which uses diagonal distance instead of traditional heuristic functions to reduce the number of turns in the path; and add turning weights in the A* algorithm formula, further reducing the number of turns in the path and reducing the number of node searches. In the process of external improvement, the output path of the internally improved A* algorithm is further optimized externally by the improved forward search optimization algorithm and the Bessel curve method, which reduces path length and turns and creates a path with fewer turns and a shorter distance. The experimental results demonstrate that the internally modified A* algorithm suggested in this research performs better when compared to six conventional path planning methods. Based on the internally improved A* algorithm path, the full improved A* algorithm reduces the turning angle by approximately 69% and shortens the path by approximately 10%; based on the simulation results, the improved A* algorithm in this paper can reduce the running time of AGV and improve the logistics efficiency in the workshop. Specifically, the walking time of AGV on the improved A* algorithm path is reduced by 12s compared to the traditional A* algorithm.

Keywords: improved A* algorithm; AGV; path planning; bezier curve; path optimization

1. Introduction

Sensible logistics are essential for advancing the manufacturing industry in the context of intelligent manufacturing. Intelligent logistics achieves process automation and intelligence by fusing

cutting-edge technologies like the Internet with intelligent instruments like smart robots, in line with the objectives of intelligent manufacturing. Modern technologies like the Internet of Things, cloud computing, big data, and artificial intelligence are essential to the smart logistics distribution system. In order to ensure seamless job completion and minimize the need for human and material resources, they expedite the design of order tasks, material distribution routes, vehicle scheduling, and load capacity. This all-encompassing strategy significantly improves production and logistical efficiencies [1]. Both software and hardware are needed to construct a smart logistics system. Stackers, different conveying equipment, and automated guided vehicles (AGV) are foundational elements. The enhancement of intelligent logistics is ensured by the digital workshop control system, path planning, navigation algorithm, and optimization algorithm.

Studies show that throughout the whole product production process, transportation, loading and unloading, and other links use three quarters of the time, while production and processing take up one quarter. Forklifts and other equipment are used in traditional transportation methods, which are dangerous, labor-intensive, and prone to mistakes. Deploying robots or AGVs as a substitute is beneficial in places with high labor costs, scarce human resources, or hazardous situations. As a component of intelligent logistics, AGV automates and simplifies the logistics transportation process, lowering labor costs in warehouses and enhancing operational effectiveness. This is in accordance with the trend of building intelligent and automated logistic systems in industrial development.

AGVs have taken on a variety of complex responsibilities in recent years. The main goal of material distribution using AGVs is to minimize human interaction by guaranteeing efficient conveyance to the work site. Therefore, solving the AGV path planning problem in the intelligent distribution system is necessary to realize intelligent logistics in the digital workshop [2,3].

AGV path planning has drawn a lot of interest due to its widespread applicability. The Dijkstra algorithm, A* algorithm, deep reinforcement learning algorithm, and ant colony algorithm are some of the most widely used path planning techniques. Based on the Dijkstra algorithm, Zhou and Huang [4] presented an improved method that uses the ant colony optimization strategy to optimize AGV pathways in airport situations. This model takes obstacle-filled situations into account. Results from simulations demonstrate that the combined algorithm in the airport baggage check-in path planning model performs better than three different methods. It is noteworthy that it reduces path lengths by 2.3, 2.64 and 6.06%. In order to provide collision avoidance when crossing known moving obstacles, He [5] and colleagues developed a dynamic search mechanism within the DAA (Dynamic A* Algorithm) that takes temporal considerations into account. The results of the simulations show that the DAA star algorithm performs better than the dynamic A* algorithm and the conventional A* algorithm in complex navigation scenarios. In order to learn the best decisions, Gu [6] and his associates used the Munchausen deep Q-learning network (m-DQN) for mobile robot applications. The simulation findings highlight how this approach learns more effectively and converges more quickly than DQN, Dueling DQN, and m-DQN counterparts, especially in static and dynamic situations. The technique is also very good at creating pathways that avoid collisions and go around obstructions. A group of researchers led by Miao [7] presented an improved adaptable ant colony algorithm (IaACO). Initially, angle guidance and obstacle removal parameters were integrated into the traditional ant colony algorithm (ACO) to increase the transmission probability. Second, adaptive pheromone volatilization factors and heuristic information adaptive adjustment were included to improve the pheromone update algorithms. The results of the experiment show that IaACO allows the robot to achieve global optimal pathways, exhibiting excellent planning stability and real-time performance.

In digital manufacturing workshops, the higher computer system must quickly determine an appropriate path for the AGV in order to guarantee the effectiveness of AGV logistics. Unfortunately, the training procedure for algorithms like the ant colony algorithm, DQN, and others takes a long time [8–12], which makes it difficult to increase the workshop's AGV's logistical efficiency. The Dijkstra algorithm wastes time by having to search through an excessive number of nodes in the search path. As a result, the A* algorithm is selected as the fundamental algorithm in this article.

The frequency of AGV starts and pauses, as well as the amount of time they spend navigating turns on the path, can both be greatly reduced with a streamlined path. This optimization is essential to increasing the overall effectiveness of logistics. Five spiral transition curve templates with monotonic (increasing or decreasing) curvature and cubic Bezier curve configurations were presented by Bibi et al. [13]. The benefits of cubic Bezier spiral path smoothing technology were illustrated through experimental validation. To tackle the issue of excessive nodes and spikes during path design, Duraklı [14] presented a novel solution based on Bezier curves. The results of the experiments demonstrated the effectiveness of the strategy in determining the best routes between the beginning and target points in well-known areas. Global smooth path planning for mobile robots with motion constraints was examined by Song [15]. They developed a method that combines spline curves and an improved particle swarm optimization technique. The efficiency of this strategy was demonstrated decisively by the outcomes of the experiment.

There are still a number of issues with path smoothing, despite the large amount of research on the subject. Interestingly, a great deal of current path smoothing research projects unintentionally introduce curvature to the straight regions of the path by applying smoothing algorithms to such segments. Once smoothed, this curvature becomes detrimental to the best possible operating of AGVs. Therefore, there is a strong need to present a novel method of path smoothing that successfully takes care of this issue.

To sum up, this post will enhance the A* algorithm on the inside as well as the outside. In order to increase the efficiency of the A* algorithm, the traditional heuristic function is first replaced with a diagonal heuristic function, and the turning cost is increased. Afterwards, redundant turns and path optimization at corners are reduced by enhancing the forward search algorithm and the Bezier curve method outside of the A* algorithm.

The main contributions of this paper are as follows:

1) Enhance the A* algorithm internally. We improve node traversal logic to stop pathways from intersecting barriers diagonally, which would produce erroneous paths. The search efficiency of the algorithm can be increased by using a diagonal distance heuristic function for 8 neighborhood search directions, which can more precisely estimate the expected cost from the current node to the target node. We reduce the number of nodes traveled during the path search of the A* algorithm and increase search efficiency by adding turning weights to the method's formula.

2) On the output path of the internally improved A* algorithm, we do external optimization. The forward search optimization technique is improved and a new algorithm is suggested in this paper. This technique can identify important turning points in the path and eliminate unnecessary ones. To smooth out important turning points, this article uses an optimization technique based on Bessel curves.

2. Problem statement and environmental modeling

2.1. Problem statement

A digital manufacturing workshop serves as the path planning algorithm's application scenario in this article. Inside the manufacturing workshop, there are numerous production scenarios, unlike in ports, airports, and storage facilities. AGVs can go straight from the conclusion of one task to the beginning of the next, eliminating the requirement for them to stop and return to a fixed node while in operation. As a result, the AGVs' starting point and destination in the production workplace are dynamic, and they must be able to swiftly plan their route from one task node to the next.

2.2. Environment

The manufacturing areas in the workshop have a basic layout, and that is where the impediments are positioned. For map modeling, this tutorial opts to utilize the grid map method. The area where an AGV may walk is shown by the white area in the grid map (Figure 1), while the area with obstacles is represented by the black area. The grid's numbering is represented by the numbers on the grid, which are written as $1,2,3,...,w$ from top to bottom and left to right. Arrows in eight directions grant authorization to search for paths from those eight directions. Specifically, the positive X-axis direction is defined from left to right, and the positive Y-axis direction is specified from bottom to top, starting from the bottom left corner of the grid area. Define the grid length as the unit length to establish a two-dimensional coordinate plane XOY [16]. The following is the correspondence between the grid number and coordinates:

$$\begin{cases} x = \mathrm{mod}(W, N) - 0.5, x > 0 \\ x = N - 0.5, x < 0 \\ y = N + 0.5 - ceil(W / N) \end{cases} \tag{1}$$

In this formula, x, y represents the horizontal and vertical coordinates of the grid map, mod is a remainder operation, W is the number of the grid, N is the number of rows and columns in the grid, and $ceil$ is the rounding function.

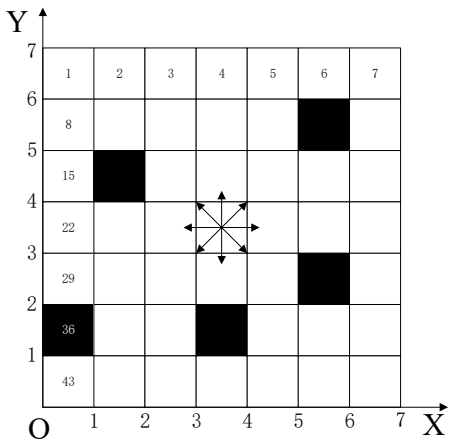


Figure 1. Grid map.

3. Introduction of A* algorithm

3.1. A algorithm*

The A* algorithm [19] constitutes a heuristic search-driven routing algorithm, designed for identifying the shortest path between a given starting point and an end point within a graph (or grid) structure. This approach employs an evaluation function to assess potential pathways and subsequently opts for the most promising route based on the outcome of the evaluation. Equation (2) delineates the evaluation function employed by the A* algorithm.

$$f(n) = g(n) + h(n) \tag{2}$$

where $f(n)$ is the total cost of current node n, $g(n)$ is the actual cost from the starting point to node n, and $h(n)$ is the estimated cost from node n to the end point.

The algorithm maintains an open list that contains the nodes to be evaluated and a closed list that contains the evaluated nodes. The flow chart of the A* algorithm is shown in Figure 2.

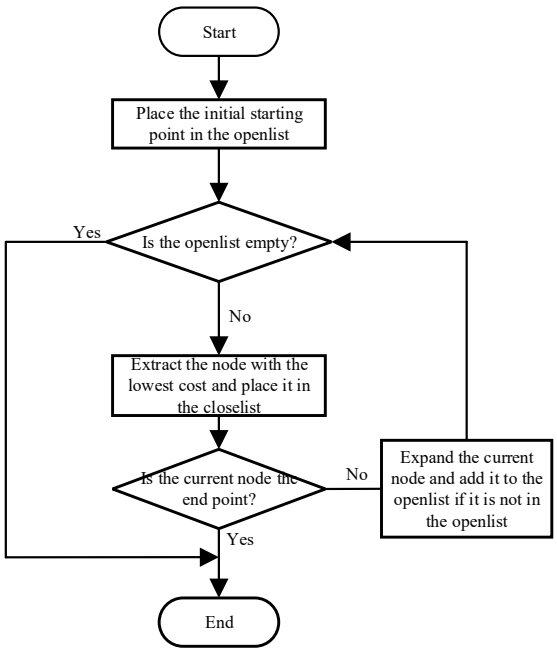


Figure 2. A* algorithm flow chart.

At present, the Euclidean distance, Manhattan distance, and Chebyshev distance are commonly used in the A* algorithm. The Euclidean distance is expressed as

$$h_o = \sqrt{(x_e - x_n)^2 + (y_e - y_n)^2} \tag{3}$$

The Manhattan distance is expressed as

$$h_m = |x_e - x_n| + |y_e - y_n| \tag{4}$$

The Chebyshev distance is expressed as

$$h_c = \max(|x_e - x_n|, |y_e - y_n|) \tag{5}$$

where (x_e, y_e) is the endpoint coordinate, and (x_n, y_n) is the current node coordinate.

3.2. Selection of actual cost function

Given that this paper designates the AGV's movement direction on the map as comprising 8 directions, both straight-line and diagonal motions of the AGV are factored into consideration. Specifically, this study presumes the cost of straight-line motion as 1 and the cost of diagonal motion as 1.4, expressed mathematically as

$$g(n) = \begin{cases} 1 & , x_n = x_{n+1} \text{ or } y_n = y_{n+1} \\ 1.4 & , \text{other} \end{cases} \tag{6}$$

where (x_n, y_n) is the coordinates of the current node and (x_{n+1}, y_{n+1}) is the coordinates of the child nodes of the current node.

3.3. Selection of estimated cost function

The estimated cost significantly influences the search direction of the A* algorithm. An accurate estimated cost effectively gauges the expense from the present node to the destination, thereby steering the A* algorithm toward a correct search direction. The trajectories corresponding to three distinct estimated costs are illustrated in Figure 3.

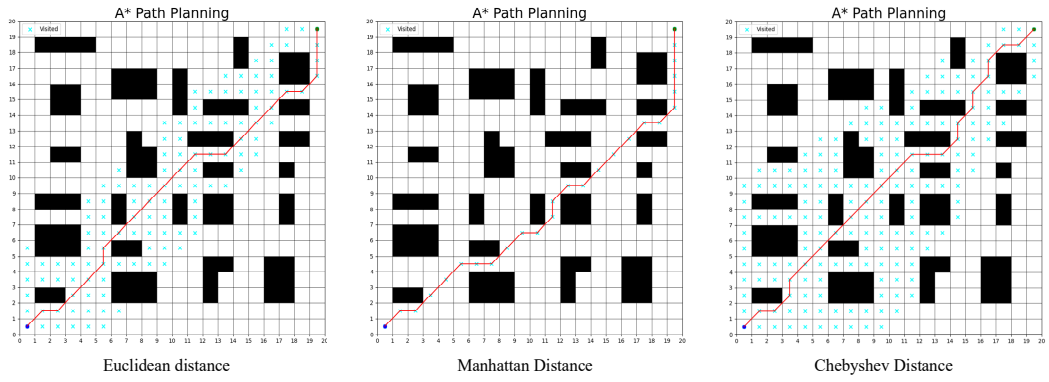


Figure 3. Comparison of paths under different estimation functions.

The evaluation criteria for the estimated cost function encompass the actual path cost, the count of turns, and the number of traversed nodes. As depicted in Figure 1, when employing Euclidean distance as the metric, the path's actual cost totals 30, involving 9 turns and traversing 117 nodes. Utilization of the Manhattan distance results in the algorithm primarily navigating through nodes during its search. Due to the sparse number of traversed nodes, the path derived from this estimation

function might not necessarily represent the optimal trajectory. Specifically, the path under Manhattan distance exhibits an actual cost of 31.2, accompanied by 13 turns and 26 traversed nodes. When employing Chebyshev distance, the path incurs the highest tally of turns and traversed nodes, at 14 turns and 172 nodes, with an actual cost of 30. A comprehensive comparison of evaluation metrics under the three distinct estimated cost functions is detailed in Table 1.

Table 1. Comparison of results.

$h(n)$	Actual cost	Number of turns	traversal nodes
Euclidean distance	30	9	117
Manhattan distance	31.2	13	26
Chebyshev distance	30	14	172

To sum up, this paper selects Euclidean distance as the estimated cost function when improving the A* algorithm.

4. Improvement of the A* algorithm

4.1. Internal improvement of A algorithm*

1) Optimize node traversal logic. In the conventional employment of the A* algorithm for path planning, the algorithm stores the traversed nodes within a closed list. Within this list, any two nodes that share a relationship are regarded as interconnected. However, this approach can lead to an issue wherein the final planned path navigates between the vertices of two obstacles, as visually demonstrated in Figure 4. Consequently, the planned path might prove impractical for real-world applications.

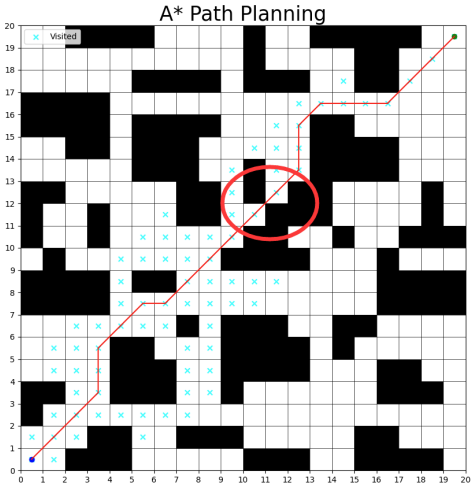


Figure 4. Schematic diagram of the path passing through the middle of the obstacle.

To address the aforementioned issues, this study enhances the node traversal logic within the traditional A* algorithm. The function Node_traversal() is constructed. When the A* algorithm is traversing the nodes, it judges whether the upper, lower, left, and right adjacent nodes of the visited

node (i, j) are obstacles. When the upper and right adjacent nodes of the node are obstacles, it skips traversing the adjacent node $(i+1, j+1)$ in the upper right direction of the node and does not put it into the close list, so as to avoid the nodes. This adjustment prevents scenarios where paths inadvertently pass through the interstices between two obstacles located at opposing corners, as depicted in Figure 5. Under the refined node traversal logic, as the traversal progresses to node (2.5, 1.5), the presence of obstacles above and to the right of this node prevents the inclusion of point (3.5, 2.5) in the closed list.

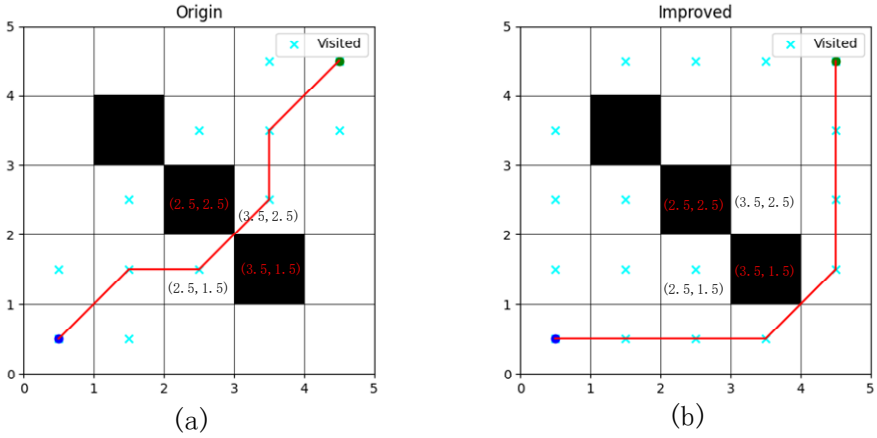


Figure 5. Schematic diagram of improved node traversal logic.

The pseudo code of function Node_traversal() is shown in Algorithm 1.

Algorithm 1

```
1. Direction = [(0, 1), (0, -1), (1, 0), (-1, 0)]
2. Current = (i,j)
3. For each direction in Direction do
4.     Neighbor = Current + each direction
5.     If up_ Neighbor = obstacle and right_ Neighbor = obstacle do
6.         Pass
7.     If up_ Neighbor = obstacle and left_ Neighbor = obstacle do
8.         Pass
9.     If down_ Neighbor = obstacle and right_ Neighbor = obstacle do
10.        Pass
11.    If down_ Neighbor = obstacle and left_ Neighbor = obstacle do
12.        Pass
13. End
```

In the pseudo code, the initial line defines the four cardinal directions: up, down, left, and right. The second line designates the presently traversed node, referred to as "current". Subsequently, lines 3 and 4 establish a loop to sequentially access the four adjacent nodes to the current node through the iterative process. Lines 5 and 6 are formulated to avoid traversing the upper-right node if both the upper and right sides of the current node are obstructed. Similarly, lines 7 and 8 are designed to bypass

traversal of the upper-left node in scenarios where the upper and left sides of the current node are obstructed. Analogously, lines 9 and 10, as well as lines 11 and 12, serve to prevent traversal of the lower-right and lower-left nodes, respectively, under conditions where the corresponding sides of the current node are occupied by obstacles.

The improved A* algorithm successfully mitigates the challenge of navigating through the vertices within obstacles, thereby enabling the A* algorithm to formulate a more rational path. The operational outcome of the improved A* approach is visually illustrated in Figure 6. Notably, as demonstrated in the figure, when the node's traversal reaches the coordinates (10.5, 11.5), the search does not encompass a node located at (11.5, 12.5).

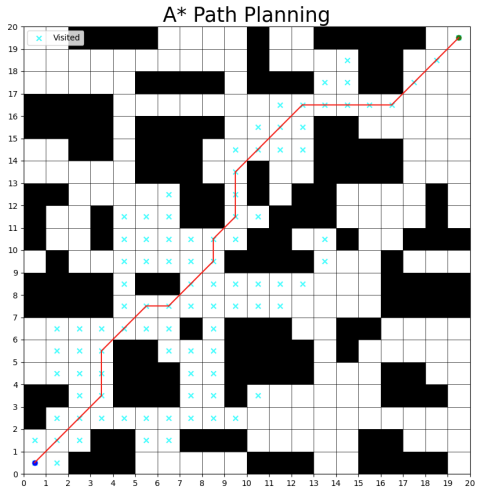


Figure 6. Algorithm running results of improved node traversal logic.

2) Replacing the conventional cost estimation function. While the A* algorithm utilizing Euclidean distance can find the shortest path with fewer traversed nodes, it might result in a higher number of turns within the path, as illustrated in Figure 7.

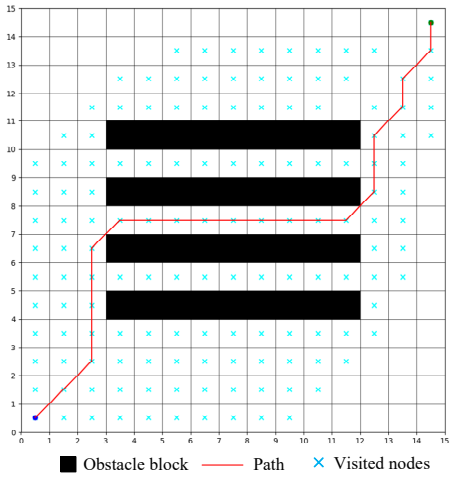


Figure 7. Path under euclidean distance.

Under such circumstances, the smooth operation of the AGV would be compromised. Therefore, this study replaces the Euclidean distance with diagonal distance in order to achieve a more accurate representation of the cost estimation function from the current point to the destination. This enhancement better guides the A* algorithm in finding the optimal path. The formula for diagonal distance is as follows:

$$h_d(n) = \min(|x_n - x_e|, |y_n - y_e|),$$

$$h_s(n) = |x_n - x_e| + |y_n - y_e|, \tag{7}$$

$$h_{total}(n) = D_1 \times h_d + D_2 \times (h_s - 2 \times h_d),$$

Among these, h_d represents the cost in the diagonal direction, h_s represents the cost in the straight line direction, (x_n, y_n) represents the current node coordinate, (x_e, y_e) represents the target node coordinate, h_{total} represents the diagonal distance calculation formula, and D_1 , D_2 represent parameters, where D_1 is 1 and D_2 is 1.4.

Upon employing diagonal distance, the planned path effectively reduces the count of redundant turns and the number of traversed nodes, as depicted in Figure 8.

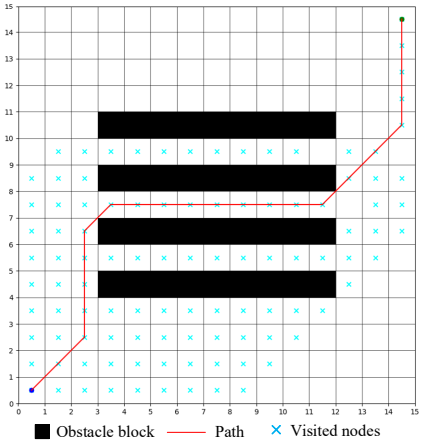


Figure 8. Paths under diagonal distance.

3) Increase the cost of turning. The conventional A* algorithm predominantly employs length as the primary criterion, prioritizing the attainment of the shortest overall distance as the optimal path. However, an exclusive focus on minimizing length may result in frequent turns by the AGV during its movement. In practical scenarios, the act of acceleration and deceleration during turning substantially escalates the time expense, rendering it considerably higher than that incurred during linear motion. This divergence in time costs undermines overall efficiency and hampers the attainment of a smooth path. To address this concern, this paper introduces an enhancement to the calculation formula of the conventional A* algorithm. This enhancement entails the incorporation of a turning cost into the original calculation formula. By introducing this turning cost, the aim is to curtail the frequency of

turns within the path, thereby fostering a more streamlined trajectory.

When accounting for the turning cost, it becomes imperative to initially assess the occurrence of a turn. In this investigation, the analysis employs the three-point collinear method for turn detection. This method involves utilizing three nodes and applying Eq (8) to perform the calculation.

$$K = (x_n - x_{n-1}) \times (y_{n+1} - y_n) - (y_n - y_{n-1}) \times (x_{n+1} - x_n) \tag{8}$$

where (x_{n-1}, y_{n-1}) is the parent node of the current node, (x_n, y_n) is the current node, and (x_{n+1}, y_{n+1}) is the current node sub node. The node relationship is shown in Figure 9.

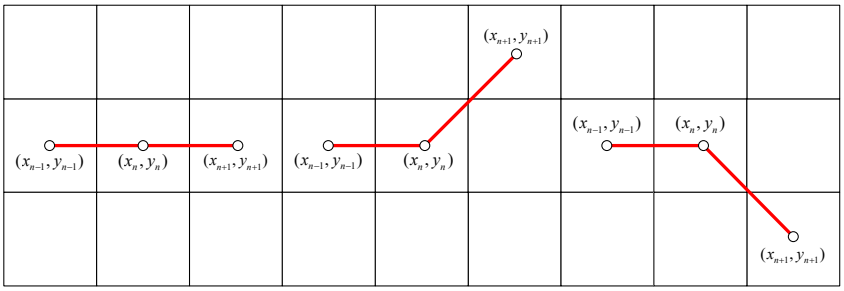


Figure 9. Node relationship.

For a straight-line path, Eq (8) yields a K value of 0. However, during turning maneuvers, distinct K values emerge. Specifically, based on the outcomes of Eq (8), a left turn corresponds to a K value of -1, while a right turn corresponds to a K value of 1. In this study, the costs associated with left and right turns are not individually considered, but rather are collectively regarded as turning costs. Consequently, the expression for the C value is illustrated in Eq (9).

$$C = \begin{cases} 1 & , K = 0 \\ 1.2 & , others \end{cases} \tag{9}$$

where C is the turning cost.

To sum up, the improved A* formula is:

$$f(n) = g(n) + C \times h_{total}(n) \tag{10}$$

Under these conditions, the A* algorithm with improved heuristic function will give priority to traversing nodes in the current direction during the node search process, which will minimize the exploration of nodes in other directions. In Figure 10, the concept is displayed. Figure 10(a),(b) depict the search procedure using conventional heuristic functions and weighted heuristic functions, respectively. It is evident that node (2.5, 1.5) determines which child nodes are selected differently between the two. When choosing child nodes, weighted heuristic functions incur a higher cost at point (2.5, 2.5) since (2.5, 1.5) is the turning point, whereas traditional heuristic functions execute a turning point at node (2.5, 1.5). Therefore, the weighted heuristic functions continue to search in the original

direction, reducing the number of turning points and traversing nodes in the path.

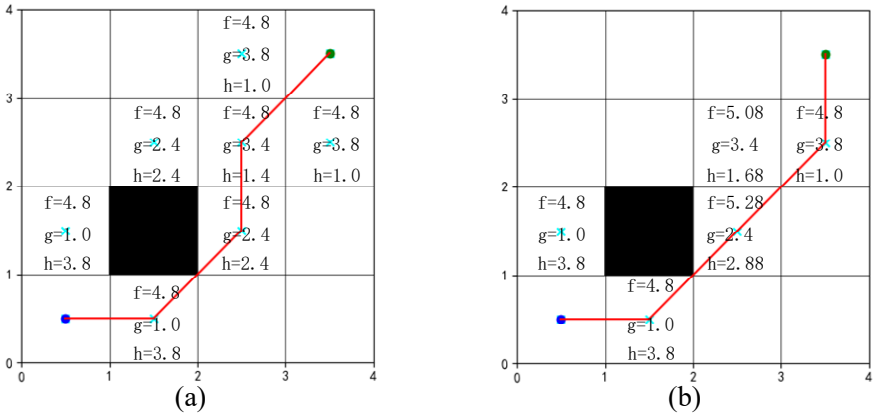


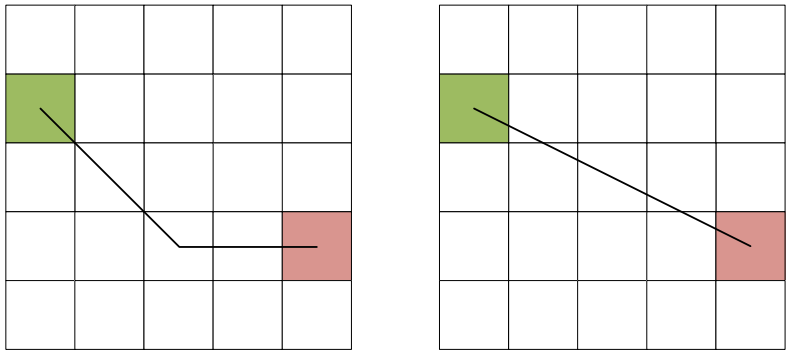
Figure 10. Weighted heuristic function search process.

4.2. External improvement of the A algorithm*

The objective of this section is to conduct further path optimization based on the output path generated by the A* algorithm. This process is chiefly divided into two primary steps: Path Optimization: Initially, the path is optimized to eliminate extraneous turning points present within it; Bezier Curve Fitting: Subsequently, the remaining turning points are subjected to a fitting process using Bezier curves.

1) Path optimization

While the internal enhancements applied to the A* algorithm effectively curtail the occurrence of turns within the path, it is important to acknowledge that the path's optimality remains contingent on the constraints of the grid map. Consequently, the resultant path might not always be the absolute optimal solution, and some degree of turns may still persist. As illustrated in Figure 11, an excessive number of turns can extend the logistics duration, thereby substantially diminishing logistics efficiency within the production workshop setting.



(a) Shortest Path in Grid Maps (b) Actual shortest path

Figure 11. Grid map restrictions.

To solve this problem, this paper adopts a path optimization method based on forward search optimization [25]. The optimization process of this method is shown in Figure 12: First, define the set of input path nodes. Then, take the starting node of the path as the current vertex, search the subsequent path nodes, find the last key node k_1 connected with the starting point (connectivity means that the connection between the starting point and k_1 point does not pass through obstacles), connect the starting point with k_1, then take k_1 as the starting point, search the points after k_1 with the same logic, and set these key points to k_2, k_3 in turn. Then, insert the sampling point s_i on the path, take s_{10} (the k_2 point in the previous step) as the current key point, and check whether the connection between $s_1 - s_5$ and s_{10} will pass through obstacles. If there are none, take s_{13} (the k_3 point in the previous step) as the current key point, and check whether the connection between $s_1 - s_9$ and s_{10} will pass through obstacles. Since point s_7 is the first point that can be connected with s_{13}, s_7 is taken as the new key point k_2. Then, connect s_7 and s_{13} to form the second optimization path, and $s_1 - s_7$ is optimized by using similar methods. Finally, the final optimization path is obtained.

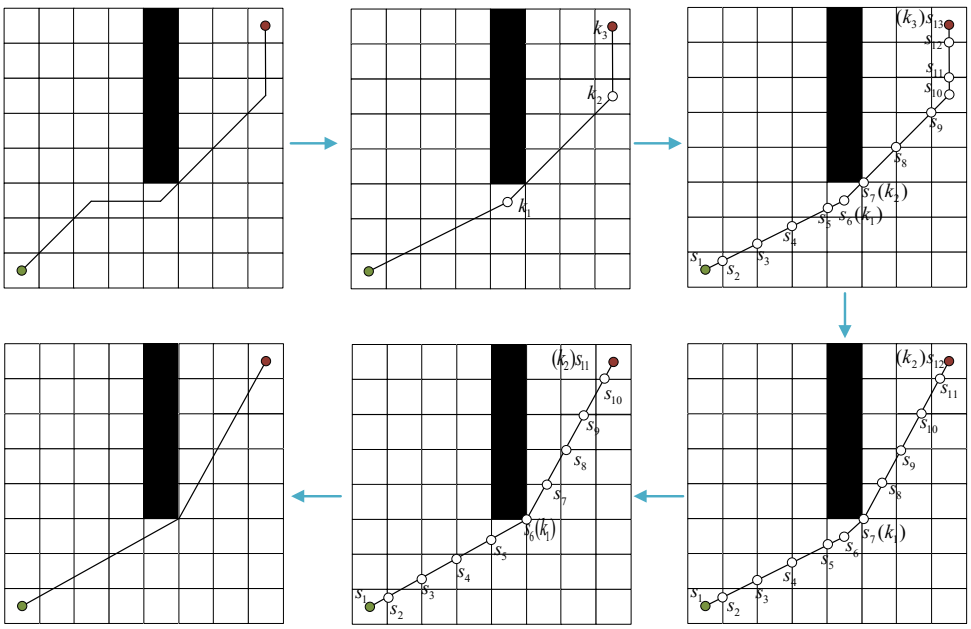


Figure 12. Process diagram of forward search optimization method.

While the concept behind the path optimization method is viable, the execution process proves excessively intricate. Consequently, this paper introduces an improved forward search optimization approach. This improved method is primarily segmented into the subsequent steps:

Step 1. The output path set $P = ([x_1, y_1], [x_2, y_2], \cdots, [x_i, y_i])$ of the A* algorithm is interpolated. The interpolation method employed here is uniform interpolation, wherein a new node is inserted between two given points. This process can be mathematically represented as

$$\begin{cases} x_{new} = \dfrac{x_{n+1} - x_n}{2} \\ y_{new} = \dfrac{y_{n+1} - y_n}{2} \end{cases} \tag{11}$$

where (x_{new}, y_{new}) are the coordinates of the inserted new node, (x_n, y_n) are the coordinates of the current node, and (x_{n+1}, y_{n+1}) are the coordinates of the child nodes of the current node.

Take the interpolated path set $P_{new} = ([x_1, y_1], [x_{new}, y_{new}], [x_2, y_2], \cdots, [x_i, y_i])$ as a new path set.

Step 2. Construct a key point set K and an obstacle vertex set Ovc. Starting from the initial point, perform a forward search to identify the first point connected to the initial point and also present in the set Ovc. Label this point as k_1 and add it to K.

Step 3. Set k_1 as the new starting point, repeat Step 2, and designate the subsequent points as $k_2, k_3, \cdots, k_i$, with i representing the number of key points.

Step 4. Repeat Step 3 to obtain the final set of key points, denoted as K.

Step 5. Connect all key points in K to achieve path optimization.

The process diagram of the improved forward search optimization method proposed in this paper is shown in Figure 13.

The pseudo code is shown in Algorithm 2

Algorithm 2

```
1: function INSERT_NEW_NODES_BETWEEN_POINTS(P)
2:     p_new←{}
3:     for i←0 to len (P)-1 do
4:         p_new.append (P [i])
5:         x1, y1←P [i]
6:         x2, y2←P [i + 1]
7:         new_node←((x1 + x2) / 2, (y1 + y2) / 2)
8:         p_new.append (new_node)
9:     end for
10:    p_new.append (P [-1])
11:    return p_new
12: end function
13: function FIND_PATH (P)
14:    p_new←insert_new_nodes_between_points (P)
15:    Ovc←[]
16:    K←[]
17:    K.append(p_new [0])
18:    start←p_new [0]
19:    for j←p_new.index (start) + 1 to len (p_new) do
20:        if line (start, p_new [j]) and p_new [j - 1]∈Ovc then
21:            K. append (p_new [j - 1])
22:            start←p_new [j - 1]
23:        end if
24:    end for
25:    K.append (p_new [-1])
26:    return K
27: end function
```

In the pseudo code of Algorithm 2, lines 1 to 12 correspond to the process in Step 1. Lines 13 to

26 correspond to the process in Steps 2–4. The $line(start, p\_new[j])$ function is used to determine whether $start$ points are connected with $p\_new[j]$ points.

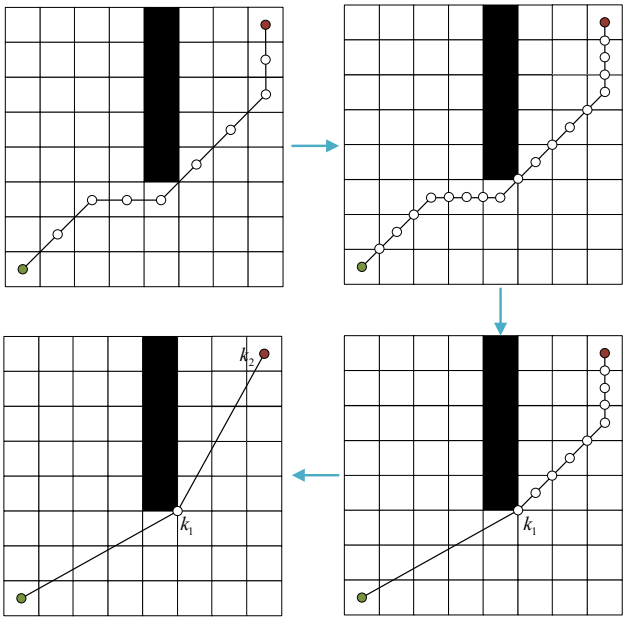


Figure 13. Process diagram of the improved forward search optimization method in this paper.

2) Path smoothing

The smoothing of the path refers to the use of Bezier curves, spline interpolation curves, and other methods to fit the path by using the curve to improve the smoothness of the path.

Although the optimized path greatly reduces turning, the turning point intersects with the vertex of the obstacle, and the AGV may collide with the obstacle when passing by. Therefore, the path must be adjusted for obstacle avoidance before curve fitting.

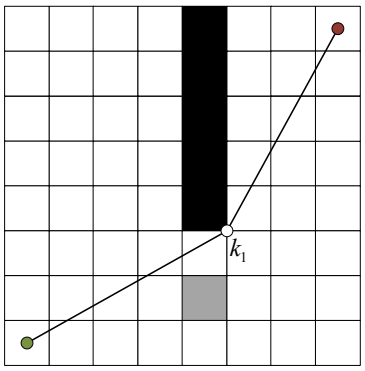


Figure 14. Schematic diagram of grid with possible obstacles around.

First, the relationship between the optimized path and the nearby obstacles is analyzed, as shown in Figure 14, where the gray grid indicates the location of the possible obstacles.

Based on this, to ensure a certain safe distance between the path and the obstacle, the obstacle avoidance adjustment process proposed in this paper is presented as follows:

Step 1. Calculate the direction vector of k_1 and obstacle coordinates according to Eq (12).

$$\vec{a} = (x_{\text{obstacle}} - x_{k_1}, y_{\text{obstacle}} - y_{k_1}) \tag{12}$$

where $\vec{a}$ is the direction vector, $(x_{\text{obstacle}}, y_{\text{obstacle}})$ is the obstacle coordinate, and (x_{k_1}, y_{k_1}) is the k_1 coordinate.

Step 2. Move one half of the grid distance in the opposite direction of k_1 along the direction vector as follows:

$$(x_{k_1\_n}, y_{k_1\_n}) = (x_{k_1}, y_{k_1}) - \frac{L}{2}\vec{a} \tag{13}$$

where $(x_{k_1\_n}, y_{k_1\_n})$ represents the new coordinates of k_1, and L represents the width of the grid.

Step 3. Update k_1 coordinates, build the linear equation from the starting point to k_1, and assess the distance between the obstacle coordinates and the straight line. When the distance is less than L, set the nearest vertex from the obstacle to the straight line to k_{insert} and repeat Steps 1 and 2. At this time, k_1 in Steps 1 and 2 is k_{insert}, and the obstacle avoidance of the path is completed. k_{insert} is located between the starting point and k_1.

Step 4. After completing the obstacle avoidance operation from the starting point to k_1, make k_1 a new starting point and repeat Steps 1 to 3 to complete the obstacle avoidance of the entire road.

The process is shown in Figure 15.

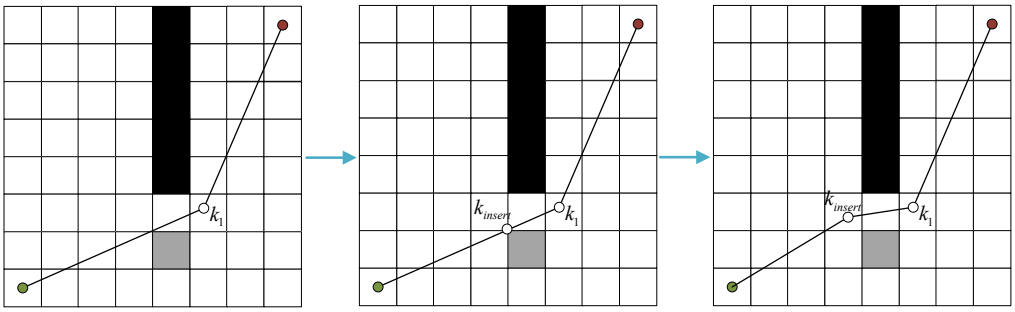


Figure 15. Path after obstacle avoidance.

Then, the second-order Bezier curve is used to fit the rotation angle. The second-order Bezier curve is mathematically expressed as

$$B(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2, t \in [0,1] \tag{14}$$

where P_0, P_1, and P_2 are the three control points, and t is a parameter.

In this paper, the selection method of control points P_0 and P_2 is explained as follows. First, calculate the direction vectors of k_{insert} and k_1 coordinates and the starting line and the next node line. Second, use a similar method in obstacle avoidance adjustment to select P_0 and P_2 control points on the two paths with length L as the distance. Finally, k_{insert} and k_1 are used as P_1 control points to achieve curve fitting at turns, as shown in Figure 16.

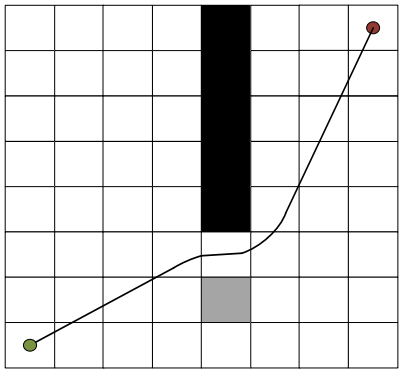


Figure 16. Path after curve fitting.

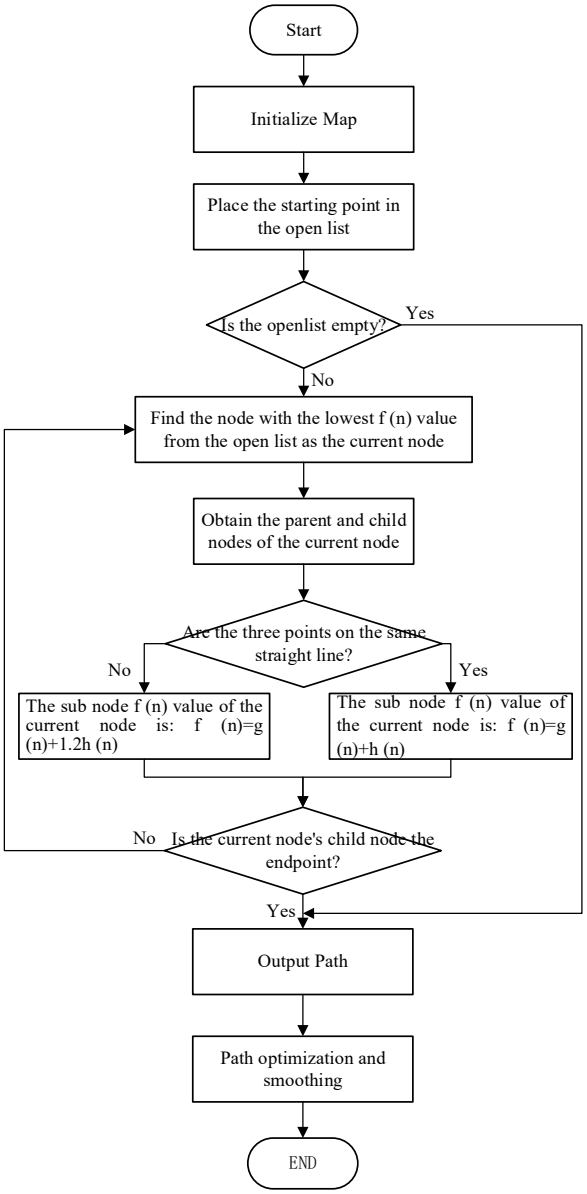


Figure 17. Flow chart of the AGV path planning method based on the improved A* algorithm.

4.3. Path planning method based on the improved A algorithm*

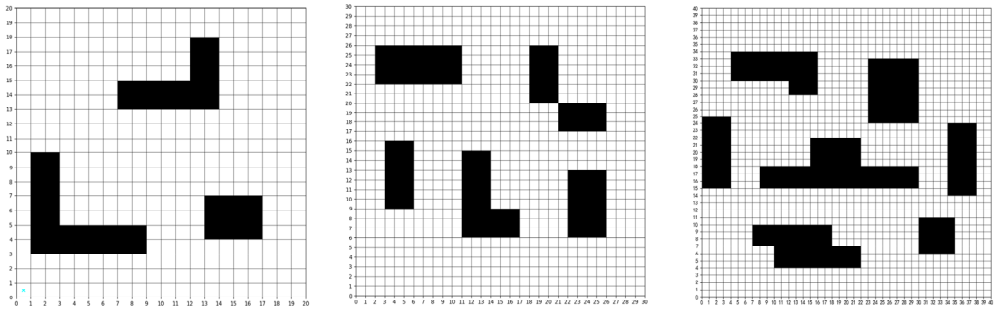
After improving the traditional A* algorithm in Sections 4.1 and 4.2, the AGV path planning method based on the improved A* algorithm can be illustrated by the flow chart in Figure 17.

5. Simulation experiments and analyses

Comprehensive experiments were conducted on the Pycharm 2022.3 platform. The programming language is Python 3.10. The performance of the proposed improved A* algorithm was tested, and the framework used by the DQN network was TensorFlow. The performance parameters of the execution host are Windows 10, and the hardware parameters are Intel(R) core(TM) i5-8250U CPU, 1.6 GHz, x64, 8 GB (RAM).

5.1. Experimental map and evaluation index

The obstacles in the production workshop are each production area. These production areas are usually large in area and simple in shape. Therefore, this article simulates and constructs maps with sizes of 20 × 20, 30 × 30, and 40 × 40 based on the characteristics of obstacles in the production workshop, as shown in Figure 18.



(a)20×20 (b)30×30 (c)40×40

Figure 18. Grid map used in the experiment.

This study uses path length, turning angle, number of traversal nodes, and algorithm running time as evaluation indices. The length of the path indicates the true cost of the path. The turning angle is the total of the degrees the AGV must turn in order to travel down the course. The number of nodes that the algorithm traverses in order to locate the path is represented by the number of traversal nodes. The algorithm's running time shows how long the path is searched throughout that period.

This paper uses both theoretical and simulated experiments for its experimental section. In the theoretical experimental part, the internal improved A* algorithm is first compared with other algorithms, and then the improved forward search optimization algorithm is verified. A custom path planner in the ROS system is used in the simulation experiment section of this article to use the traditional A* algorithm, internally improved A* algorithm, and improved A* algorithm as plugins. Simulation verification is carried out by calling algorithm plugins.

5.2. Theoretical experiments

5.2.1. Performance verification of internal improved A* algorithm

The algorithms involved in the comparison include the traditional A* algorithm, Dijkstra algorithm, bidirectional A* algorithm, ant colony algorithm, DQN algorithm, and the internal improved A* algorithm in this paper. The starting point is the (0.5, 0.5) point, and the ending point is the point in the upper right corner. The cyan and red grids in the grid graph represent the traversed nodes. The parameters of the ant colony algorithm are shown in Table 2, and the parameters of the DQN algorithm are shown in Table 3.

Table 2. Ant colony algorithm parameters.

parameter	value
Number of ants	20
Number of iterations	150
Pheromone evaporation rate	0.5
pheromone factor	1.1
heuristic function factor	4

Table 3. DQN algorithm parameters.

parameter	value
Learning rate	0.001
Discount factor	0.995
Exploration rate	0.4
Batch size	64

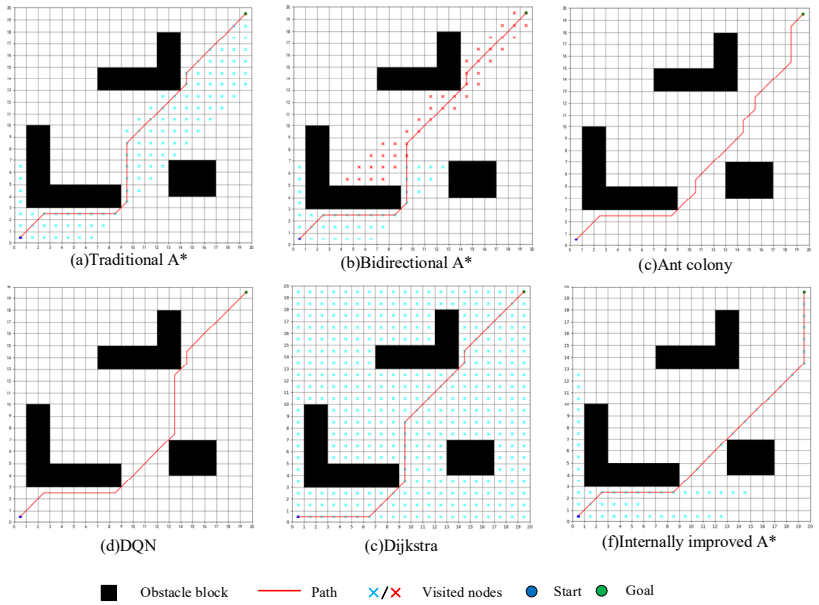


Figure 19. 20 × 20 comparison of results in the map.

As shown in Figure 19, when comparing the ant colony method to other algorithms at a 20 × 20 grid map, it yielded the longest path length, with the other algorithms obtaining the same path length. The turning angle of the traditional A*, bidirectional A*, ant colony algorithm, DQN algorithm, and Dijkstra method has grown by 135, 135, 315, 135, and 90°, respectively. The ant colony algorithm and DQN algorithm have distinct search algorithms from the others, therefore, the indicator of the number of traversal nodes is not considered, however, the traditional A* algorithm, bidirectional A* algorithm, and Dijkstra algorithm have increased by 43, 35, and 285, respectively. Algorithm running times rose by 0.51 s for the traditional A* algorithm, 0.24 s for the bidirectional A* method, 4.59 s for the ant colony algorithm, 4.33 s for the DQN algorithm, and 1.59 s for the Dijkstra algorithm.

As shown in Figure 20, the conventional A* algorithm outperforms the internal improved A* algorithm on a 30 × 30 grid map in terms of path length, number of traversed nodes, and algorithm running time. The bidirectional A* algorithm outperforms the internal improved A* algorithm in terms of both these metrics. The ant colony and DQN algorithms outperform the internal improved A* algorithm in terms of path length. Although in some indicators the comparison algorithm outperforms the internal improved A* algorithm, the comparison algorithm's planned path, which goes from nodes (20.5, 19.5) to (21.5, 20.5), diagonally passes through two obstacle nodes (20.5, 20.5) and (19.5, 21.5), making the expected path AGV invalid. This illustration confirms that the A* algorithm's internal improvement can prevent the creation of erroneous pathways.

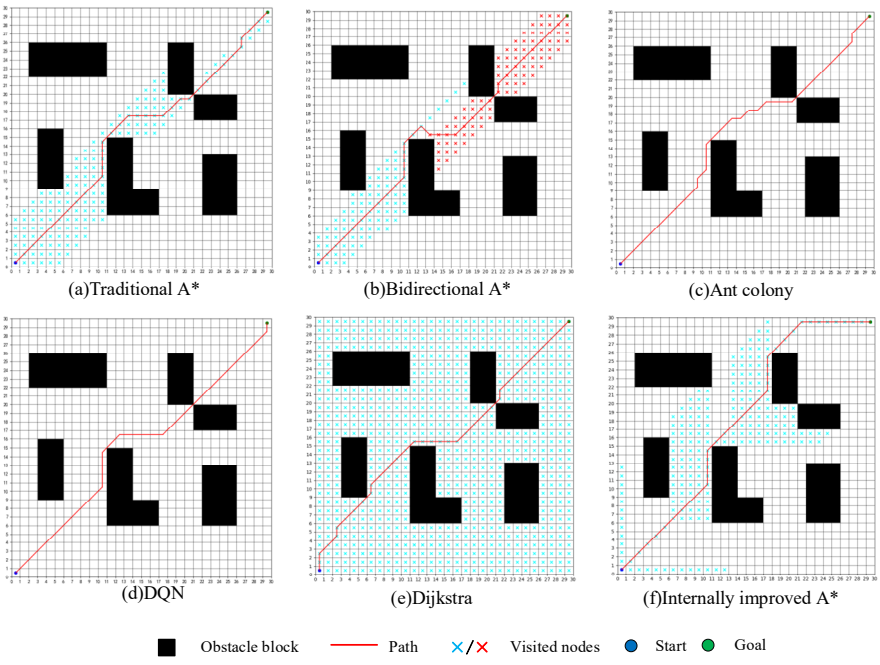


Figure 20. 30 × 30 comparison of results in the map.

As shown in Figure 21, the ant colony algorithm produced the longest path length on a 40 × 40 grid map when compared to other algorithms, and the other algorithms produced the same path length. The turning angle of the traditional A*, bidirectional A*, ant colony algorithm, DQN algorithm, and the Dijkstra algorithm has risen by 135, 315, 360, 225, and 45°, respectively. Regarding traversing the number of nodes, the traditional A* algorithm, bidirectional A* algorithm, and Dijkstra algorithm have

increased by 12, 87, and 863 respectively; however, there is no need to traverse the number of nodes because the ant colony algorithm and DQN algorithm have different search mechanisms than the others. Algorithm running times rose by 0.76 s for the traditional A* algorithm, 0.47 s for the bidirectional A* method, 10.47 s for the ant colony algorithm, 9.54 s for the DQN algorithm, and 2.05 s for the Dijkstra algorithm.

The specific experimental results of each algorithm are shown in Table 4.

Table 4. Comparison of results in different environments.

Map	Algorithm	Length	Turning angle	Number of traversal nodes	Algorithm run time
20 × 20	Traditional A*	30.2	270°	104	1.29 s
	Bidirectional A*	30.2	270°	96	1.02 s
	Ant Colony	31.6	450°	\	5.37 s
	DQN	30.2	270°	\	5.11 s
	Dijk stra	30.2	225°	346	2.37 s
	Internally improved A*	30.2	135°	61	0.78 s
30 × 30	Traditional A*	43.6	360°	151	1.35 s
	Bidirectional A*	45.4	315°	162	1.24 s
	Ant Colony	43.6	540°	\	8.32 s
	DQN	43.6	225°	\	9.43 s
	Dijk stra	43.6	405°	764	2.43 s
	Internally improved A*	45.4	225°	206	1.34 s
40 × 40	Traditional A*	60.6	270°	362	2.32 s
	Bidirectional A*	65.2	450°	437	2.03 s
	Ant Colony	68.6	495°	\	12.03 s
	DQN	65.4	360°	\	11.10 s
	Dijk stra	62.2	180°	1213	3.61 s
	Internally improved A*	60.6	135°	350	1.56 s

In summary, this comparison demonstrates the usefulness of the internal modification of the A* algorithm in this work by demonstrating the modified algorithm's good performance in terms of path length, turning angle, number of traversal nodes, and algorithm running time.

5.2.2. Path optimization and smoothness verification

This part is dedicated to path optimization and smoothing verification. The path generated by the internally enhanced A* algorithm serves as the initial path input and is subsequently smoothed and optimized. The comparative algorithms are the internally enhanced A* algorithm and the improved A* algorithm (with both external and internal enhancements). The turning angle and length of the path act as assessment marks. In Figure 22, the blue dotted line represents the initial path, the green dotted line represents the optimized path, and the red solid line represents the final smoothed path.

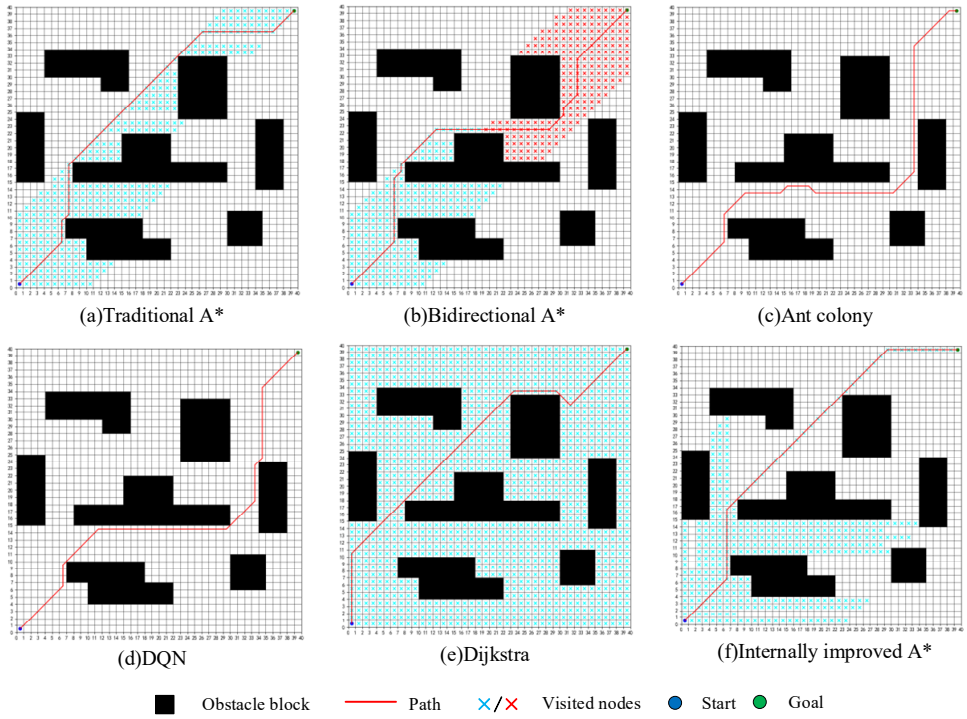


Figure 21. 40 × 40 comparison of results in the map.

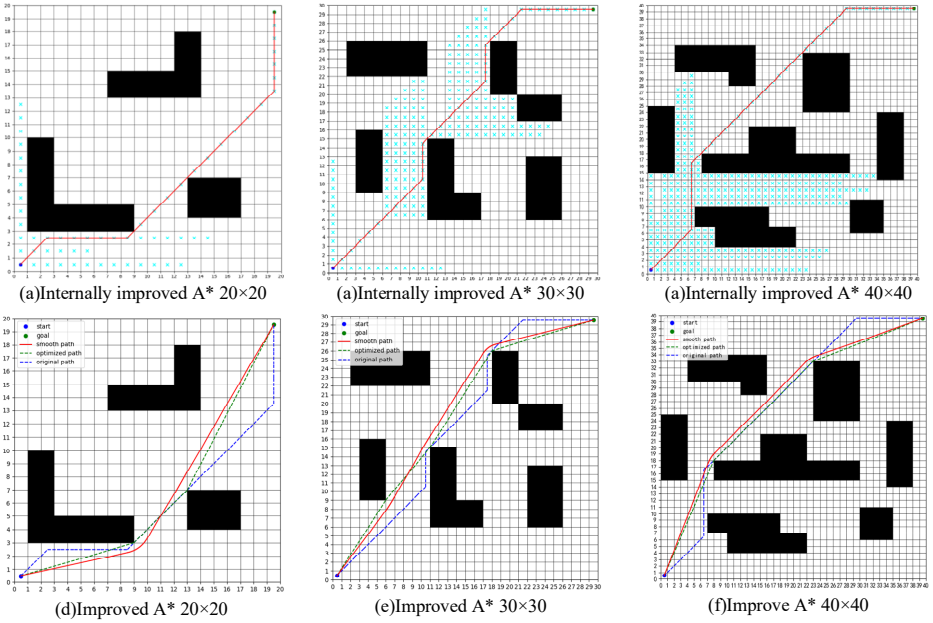


Figure 22. Path optimization and smoothing verification.

As can be seen from Table 5, based on the internal improved A* algorithm, after optimizing and smoothing the path, the path length and turning angle obtained are both shorter than the path obtained by the internal improved A* algorithm. In the 20 × 20 map, compared with the internal improved A* algorithm results, the final improved A* algorithm path length is reduced by 4.4, and the turning angle

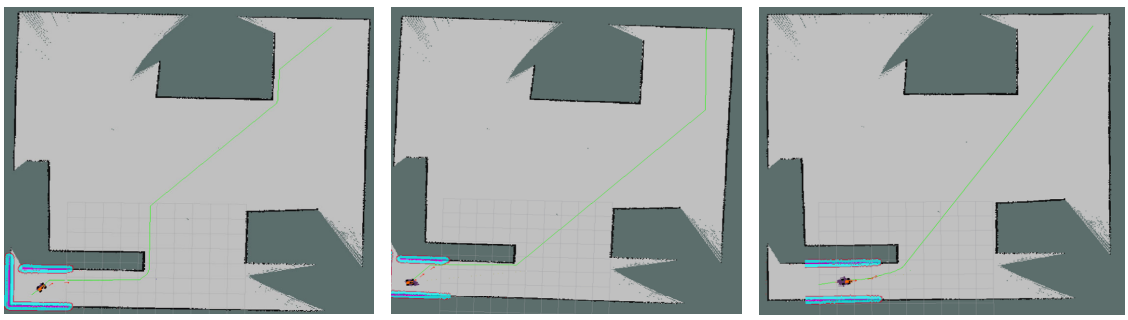
is reduced by 88.49°. In the 30 × 30 map, compared with the internal improved A* algorithm results, the final improved A* algorithm path length is reduced by 4.7, and the turning angle is reduced by 176.79°. In the 40 × 40 map, compared with the internal improved A* algorithm results, the final improved A* algorithm path length is reduced by 5.2, and the turning angle is reduced by 85.69°. It shows that the improved A* algorithm proposed in this article can effectively reduce the length of the path and the turning angle in the path, and can effectively reduce the distance and steering angle of the AGV, ensuring the smoothness of AGV driving.

Table 5. Comparison of path optimization and smoothing results.

Map	Algorithm	Length	Turning angle
20 × 20	Internally improved A*	30.2	135°
	Improved A*	25.8	46.51°
30 × 30	Internally improved A*	45.4	225°
	Improved A*	40.7	48.21°
40 × 40	Internally improved A*	60.6	135°
	Improved A*	55.4	49.31°

5.3. Simulated experiments

This article's suggested algorithm has been verified in ROS. Using the 20 × 20 map as an example, just the essential components of the map were built during the modeling of the map SLAM technique in order to conserve computing power. Figure 23 displays the simulation results. The path is represented by the green line, open space is represented by the gray section, and obstacle boundaries are displayed by the black part.



(a)Traditional A* (b)Internal improvement A* (c)Improved A*

Figure 23. Simulation results.

The paths designed by the conventional A* algorithm are depicted in Figure 23(a), the internally improved A* method in Figure 23(b), and the improved A* algorithm in Figure 23(c). The three paths take the AGV 21, 15, and 9 seconds to complete, respectively. This demonstrates that the path quality predicted by the revised A* method put forth in this research is high, which can shorten the AGV's running time and increase its effectiveness.

6. Conclusions

Although the A* algorithm has the advantages of being a simple algorithm and having a fast search speed, the planned path may involve more turns and the path may not be smooth enough. In order to solve this problem, this paper improves the internal and external properties of the traditional A* algorithm. The node traversal logic of the A* algorithm is internally optimized to avoid the path passing between two obstacle vertices. Subsequently, the cost estimation function was modified to replace the traditional Euclidean distance with the diagonal distance, thereby improving the accuracy of cost prediction. This adjustment effectively reduces the number of redundant turns within the path. Then, the weight of the cost estimation function of the A* algorithm is added to give priority to the current search direction when finding a path, thereby reducing the number of nodes traversed and the number of turns compared to the traditional A* algorithm. Finally, a path optimization smoothing method based on the improved forward search optimization method and Bezier curve method is proposed. This method is used in addition to the A* algorithm to eliminate redundant turning points in the path and perform curve fitting at turns to improve the smoothness of the path.

Finally, the experimental results show that the internal modification of the A* algorithm avoids planning invalid paths, reduces the number of turns, reduces the number of traversed nodes, and shortens the search time. After external modifications to the A* algorithm, the number of turns in the path is further reduced and the length of the path is shortened.

Although the path planning method based on the improved A* algorithm proposed in this article can effectively make up for the shortcomings of the traditional A* algorithm, it still encounters some shortcomings and defects:

1) In the actual workshop environment, the location of obstacles cannot be accurately located, but dynamic obstacles may appear, and the method proposed in this article is based on the path planning method in a static obstacle environment.

2) The improved A* algorithm proposed in this article is suitable for environments with concentrated obstacles such as workshops, but is not suitable for areas with many and scattered obstacles, such as in warehousing environments.

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

The authors thank the anonymous reviewers for their valuable comments. This work was supported by high level Project of Shihezi University (No. RCZK2021B16); The high-level project led by Professor Baoqin Wen at Shihezi University (No. RCZK202310).

Conflict of interest

The authors declare there is no conflict of interest

References

1. Z. C. Qin, Research on intelligent selection algorithm of ship logistics optimal route, *Ship Sci. Technol.*, **46** (2018), 75–86.
2. X. L. Zhao, L. W. Zhong, Optimization simulation and transport path analysis of intelligent warehouse based on flexsim, *Software Guide*, **21** (2018), 55–73.
3. M. Luo, X. R. Hou, J. Yang, Surface optimal path planning using an extended Dijkstra algorithm, *IEEE Access*, **8** (2020), 147827–147838. https://doi.org/10.1109/access.2020.3015976
4. Y. L. Zhou, N. N. Huang, Airport AGV path optimization model based on ant colony algorithm to optimize Dijkstra algorithm in urban systems, *Sustainable Comput. Inf. Syst.*, **35** (2022), 100716. https://doi.org/10.1016/j.suscom.2022.100716
5. Z. B. He, C. G. Liu, X. M. Chu, R. R. Negenborn, Q. Wu, Dynamic anti-collision A-star algorithm for multi-ship encounter situations, *Appl. Ocean Res.*, **118** (2022), 102995. https://doi.org/10.1016/j.apor.2021.102995
6. Y. W. Gu, Z. T. Zhu, J. D. Lv, L. Shi, Z. J. Hou, S. K. Xu, DM-DQN: Dueling Munchausen deep Q network for robot path planning, *Complex Intell. Syst.*, **9** (2023), 1–14. https://doi.org/10.1007/s40747-022-00948-7
7. C. W. Miao, G. Z. Chen, C. L. Yan, Y. Y. Wu, Path planning optimization of indoor mobile robot based on adaptive ant colony algorithm, *Comput. Ind. Eng.*, **156** (2021), 107230. https://doi.org/10.1016/j.cie.2021.107230
8. S. Y. Guo, X. G. Zhang, Y. Q. Du, Y. S. Zheng, Z. Y. Cao, Path planning of coastal ships based on optimized DQN reward function, *J. Mar. Sci. Eng.*, **9** (2021), 210. https://doi.org/10.3390/jmse9020210
9. T. Wang, Z. L. Xue, X. Q. Dong, S. L. Xie, Autonomous intelligent planning method for welding path of complex ship components, *Robotica*, **39** (2021), 428–437. https://doi.org/10.1017/s0263574720000454
10. Z. H. Han, S. G. Liu, F. Yu, X. D. Zhang, G. X. Zhang, A 3D measuring path planning strategy for intelligent CMMs based on an improved ant colony algorithm, *Int. J. Adv. Manuf. Technol.*, **93** (2017), 1487–1497. https://doi.org/10.1007/s00170-017-0503-y
11. N. Saito, T. Oda, A. Hirata, Y. Nagai, M. Hirota, K. Katayama, et al., A Tabu list strategy based DQN for AAV mobility in indoor single-path environment: implementation and performance evaluation, *Internet Things*, **14** (2021), 100394. https://doi.org/10.1016/j.iot.2021.100394
12. W. J. Meng, Q. Zheng, L. Yang, P. F. Li, G. Pan, Qualitative measurements of policy discrepancy for return-based deep q-network, *IEEE Trans. Neural Networks Learn. Syst.*, **31** (2019), 4374–4380. https://doi.org/10.1109/tnnls.2019.2948892
13. S. BiBi, M. Y. Misro, M. Abbas, Smooth path planning via cubic GHT-Bézier spiral curves based on shortest distance, bending energy and curvature variation energy, *AIMS Math.*, **6** (2021), 8625–8641. https://doi.org/10.3934/math.2021501
14. Z. Durakli, V. Nabiyev, A new approach based on Bezier curves to solve path planning problems for mobile robots, *J. Comput. Sci.*, **58** (2022), 101540. https://doi.org/10.1016/j.jocs.2021.101540
15. B. Y. Song, Z. D. Wang, L. Zou, L. Xu, F. E. Alsaadi, A new approach to smooth global path planning of mobile robots with kinematic constraints, *Int. J. Mach. Learn. Cybern.*, **10** (2019), 107–119. https://doi.org/10.1007/s13042-017-0703-7

16. X. Li, L. Wang, Application of improved ant colony optimization in mobile robot trajectory planning, *Math. Biosci. Eng.*, **17** (2020), 6756–6774. https://doi.org/10.3934/mbe.2020352
17. K. Fransen, J. van Eekelen, Efficient path planning for automated guided vehicles using A* (Astar) algorithm incorporating turning costs in search heuristic, *Int. J. Prod. Res.*, **61** (2023), 707–725. https://doi.org/10.1080/00207543.2021.2015806
18. B. Wu, X. N. Chi, C. C. Zhao, W. Zhang, Y. Lu, D. Jiang, Dynamic path planning for forklift AGV based on smoothing A* and improved DWA hybrid algorithm, *Sensors*, **22** (2022), 7079. https://doi.org/10.3390/s22187079
19. P. E. Hart, N. J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, *IEEE Trans. Syst. Sci. Cybern.*, **4** (1968), 100–107. https://doi.org/10.1109/tssc.1968.300136
20. A. K. Guruji, H. Agarwal, D. K. Parsediya, Time-efficient A* algorithm for robot path planning, *Procedia Technol.*, **23** (2016), 144–149. https://doi.org/10.1016/j.protcy.2016.03.010
21. X. D. Wang, H. W. Zhang, S. Liu, J. L.Wang, Y. H. Wang, D. H. Shangguan, Path planning of scenic spots based on improved A* algorithm, *Sci. Rep.*, **12** (2022), 1320. https://doi.org/10.1038/s41598-022-05386-6
22. X. L. Tong, S. E. Yu, G. Y. Liu, X. D. Niu, C. J. Xia, J. K. Chen, A hybrid formation path planning based on A* and multi-target improved artificial potential field algorithm in the 2D random environments, *Adv. Eng. Inf.*, **54**(2022), 101755. https://doi.org/10.1016/j.aei.2022.101755
23. C. G. Li, X. Huang, J. Ding, K. Song, S. Q. Lu, Global path planning based on a bidirectional alternating search A* algorithm for mobile robots, *Comput. Ind. Eng.*, **168** (2022), 108123. https://doi.org/10.1016/j.cie.2022.108123
24. C. W. Miao, G. Z. Chen, C. L. Yan, Y. Y. Wu, Path planning optimization of indoor mobile robot based on adaptive ant colony algorithm, *Comput. Ind. Eng.*, **156** (2021), 107230. https://doi.org/10.1016/j.cie.2021.107230
25. H. D. Li, T. Zhao, S. Dian, Forward search optimization and subgoal-based hybrid path planning to shorten and smooth global path for mobile robots, *Knowl.-Based Syst.*, **258** (2022), 110034. https://doi.org/10.1016/j.knosys.2022.110034

AIMS Press