Research and Simulation of the A* Algorithm in Matlab

Yifei Chen *

School of Mechanical & Electrical Engineering, Heilongjiang University, Harbin, 150080, China

* Corresponding Author Email: 20231635@s.hlju.edu.cn

Abstract. With the rapid advancement of artificial intelligence and networking, path planning has found widespread applications across various domains, including artificial intelligence, mechanical control, robotics, and the automotive industry. The A* algorithm is a very common path-planning algorithm. Traditional research mainly focuses on the application of the A* algorithm and its combination with other algorithms, while neglecting the study of the A* algorithm itself, such as its heuristic function. This article will conduct an in-depth study of the A* algorithm using MATLAB, introduce the weighted A* and dynamic A* algorithm, and study the parameter influence and improvement of the A* algorithm. Research has shown that when the weight is greater than 1, it will reduce the optimality of the results; Smaller weights usually lead to better exploration, but the search process may slow down. The dynamic A* algorithm has many characteristics such as dynamic updates and reverse search. Therefore, this study is of great significance for improving the efficiency of path planning.

Keywords: Matlab; Path planning; A* algorithm; Weighted A* algorithm; Dynamic A* algorithm.

1. Introduction

Nowadays, path planning has developed in many fields, such as AI and mechanical manufacturing [1]. In the rapidly advancing environment of robotics and artificial intelligence, path planning also includes many complex factors such as uncertainty or dynamism. In AI, path planning means searching for a series of logical actions to transform the initial robot state into the desired target state. The path planning problem has a wide range of applications, such as autonomous vehicles or intelligent robots [2]. Whether or not the obstacles on the map are dynamic or static is the primary distinction between global and local path planning. Currently, the A* algorithm and rapidly random tree (RRT) are examples of sophisticated path planning techniques [3], other methods include the artificial potential field approach and the artificial moment method. Global path planning should be more successful using the paths produced by the conventional A* method.

Many scholars have conducted research on path planning. Path planning for autonomous mobile robots has been researched by Sariff et al., who have also proposed algorithms for creating the best courses for the robots to take while navigating their surroundings [4]. It has been demonstrated that suitable algorithms can operate quickly enough to be employed in practice without causing time-consuming problems; algorithms are essential in producing the best routes for autonomous robot navigation. For unmanned aerial vehicles, Boroff et al. investigated a two-step path-planning method [5]. An intuitive way to balance stealth and path length is provided by this algorithm, which generates stealth paths through a set of known enemy radar stations [5]. Wang researched an algorithm for dynamically planning a collision avoidance path using ship navigation rules and A* [6]. In multi-ship encounter scenarios, the proposed DAA star algorithm can resolve the path planning problem of dynamic obstacles by taking into account the restrictions of maneuverability for individual ships as well as the variations in maneuverability among them [6].

The A* algorithm has been the subject of much study by numerous academics. For intelligent guided vehicles, Tang et al. investigated a path-planning technique based on the geometric A star algorithm [7]. The geometric A* algorithm demonstrates its effectiveness by successfully lowering the number of nodes and the pathfinding distance; In order to address the drawbacks of conventional A* algorithms, Erke et al. investigated a novel autonomous land vehicle path planning algorithm that developed heuristic functions based on recommendations produced globally or manually [8]. Candra et al. employed the A* method to develop and evaluate pathfinding and applied it in a tree-planting

game. In addition, a novel variable step size A* algorithm was presented to shorten the computing time of the suggested algorithm [9]. The number of accessed nodes increased with the number of obstacle nodes in the grid, according to the results; Tang et al. then rasterized and optimized the A* algorithm for modeling indoor environments, and MATLAB simulation experiments were used to confirm the optimization algorithm's viability [10]. According to the experimental results, in two distinct tests with different goals, the revised method can cut memory consumption by more than 60%. To meet the performance requirements of path smoothness, response speed, and computing time for home cleaning robot path planning. Liu et al. presented a fusion path-finding algorithm based on an optimized A star algorithm, artificial potential field method, and least squares method [11]. Global path planning can be completed quickly thanks to the fusion algorithm, which enhances the conventional A* method's operating criteria.

There is a lack of improvement in the heuristic function despite the numerous changes made to the A* algorithm by researchers, suggesting that there is still a gap in the literature about the A* heuristic concept. The standard A* algorithm and Dijkstra's algorithm are first reviewed, followed by the introduction of the weight parameter w, the proposal of a variant weighted A* algorithm, and a Matlab code simulation of the algorithm's execution outcomes and associated parameters at various weights. At last, the dynamic A* algorithm is suggested and additional study on the A* algorithm is carried out. Machines can eventually tackle dynamic path problems by using the method to adjust paths and monitor environmental changes in real time.

2. Method

2.1. Traditional A* algorithm

One of the most well-known path-planning algorithms, A* can be used in topological or metric configuration spaces [12]. The search algorithm combines heuristic and shortest-path searching techniques. A heuristic search algorithm called the A* algorithm locates the shortest path in a network between the starting node and the destination node. It combines the efficiency of the greedy algorithm with the certainty of Dijkstra's algorithm. The heuristic function of the A* algorithm is as follows:

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

Where h (n) is the estimated lowest cost from vertex n to the goal, typically determined using heuristic functions that need to be specially built to represent real terrain or environmental factors, and g (n) is the actual cost from the starting point to any vertex n. The A* method guarantees that the lowest cost path is found in this scenario, where h (n) should not overestimate the real cost. The choice of heuristic functions has a significant impact on the efficiency and precision of algorithms. It is a path-planning method that was successfully tested and implemented for a mobile robot. The findings are shown in Ref [13]. The basic A* algorithm used for a grid configuration space is restricted to 8-connectivity. This means that it can find a path that is based on the connection between the closest possible cells [1].

The running diagram of the traditional A* algorithm is shown in Fig. 1.

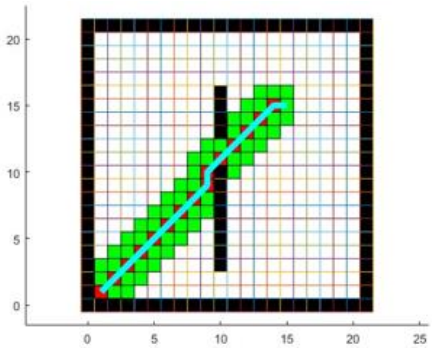

Fig 1. Traditional A* algorithm running diagram. (Photo/Picture credit: Original).

2.2. Weighted A* algorithm

A variation of the A* search algorithm, the Weighted A* algorithm modifies the search process by adding a weight factor that modifies the influence of the heuristic function below:

$$F(n) = h(n) + w * g(n). \quad (2)$$

Where h(x) heuristically guesses the cost from the current node to the target node (i.e., the estimated cost) and g(x) is the actual cost (i.e., known cost) from the start node to the current node. When w equals 2, the weighted A* algorithm runs as shown in Fig. 2.

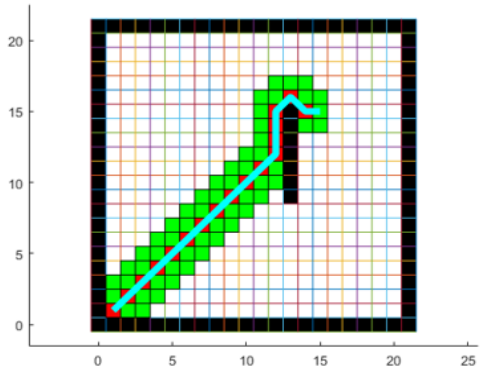

Fig 2. Weighted A* algorithm running diagram. (Photo/Picture credit: Original).

2.3. D* Algorithm

The primary application of the D* dynamic incremental path planning algorithm is to choose the best course across a dynamically changing environment. This particular A* algorithm variation allows for fast path updates without recalculating the paths of all nodes because it is optimized for changes in dynamic situations, such as the movement of barriers. During the path-planning phase, the D* method first determines the starting path using the reverse Dijkstra algorithm. This method sets the foundation for the path replanning that follows by figuring out the shortest path between the beginning location and the target point.

When the robot comes across an impediment, the D* algorithm allows it to stop in time by defining a repulsive force field. The robot can effectively navigate intricate situations by adjusting its repulsion field's strength and range dynamically based on the size and proximity of objects. The D* algorithm reroutes the path when an obstruction is found. Similar to the heuristic A* method, this procedure starts with a number of randomly distributed places, and the heuristic function h () is crucial in assisting in the selection of a new travel direction.

3. Results and Discussions

3.1. A* Algorithm

The A* algorithm is the popular name for the A* search algorithm. One of the well-liked heuristic search techniques in the field of path optimization is the A* algorithm. Its distinctiveness comes from the addition of global data during the examination of every potential node on the shortest path, the estimation of the distance between the present node and the endpoint, and the application of this distance as a metric to assess the probability that the node is on the shortest path. Table 1 is derived by comparing the A* algorithm with Dijkstra's algorithm.

Table 1. Comparison between Dijkstra's algorithm and A* algorithm.

Principle	The known shortest path is progressively extended based on the greedy method.	The heuristic function is used to evaluate nodes by combining a greedy algorithm and a heuristic search
Scope of application	An empowered graph or a weighted graph in which all side weights are positive	A complex environment with obstacles
Peculiarity	Global optimality, but high computational complexity	The search efficiency is high, and the heuristic function design affects the performance
Advantages	Can find the global shortest path	Quickly find the shortest path and adapt to complex environments
Inferior position	Computationally heavy, not suitable for large graphs	The heuristic function design is complex and may affect the result

3.2. Weighted A* Algorithm (Under the range of w=0~1)

When w=0, weighted A* degenerates to Dijkstra
When w=1, weighted A* degenerates to A*
When w=∞, weighted A* degenerates to GBFS (Greedy Best-First Search)

3.3. Weighted A* Algorithm (W is any value)

Simulate the weight A* algorithm under different weights using MATLAB, the running results are shown in Fig. 3.

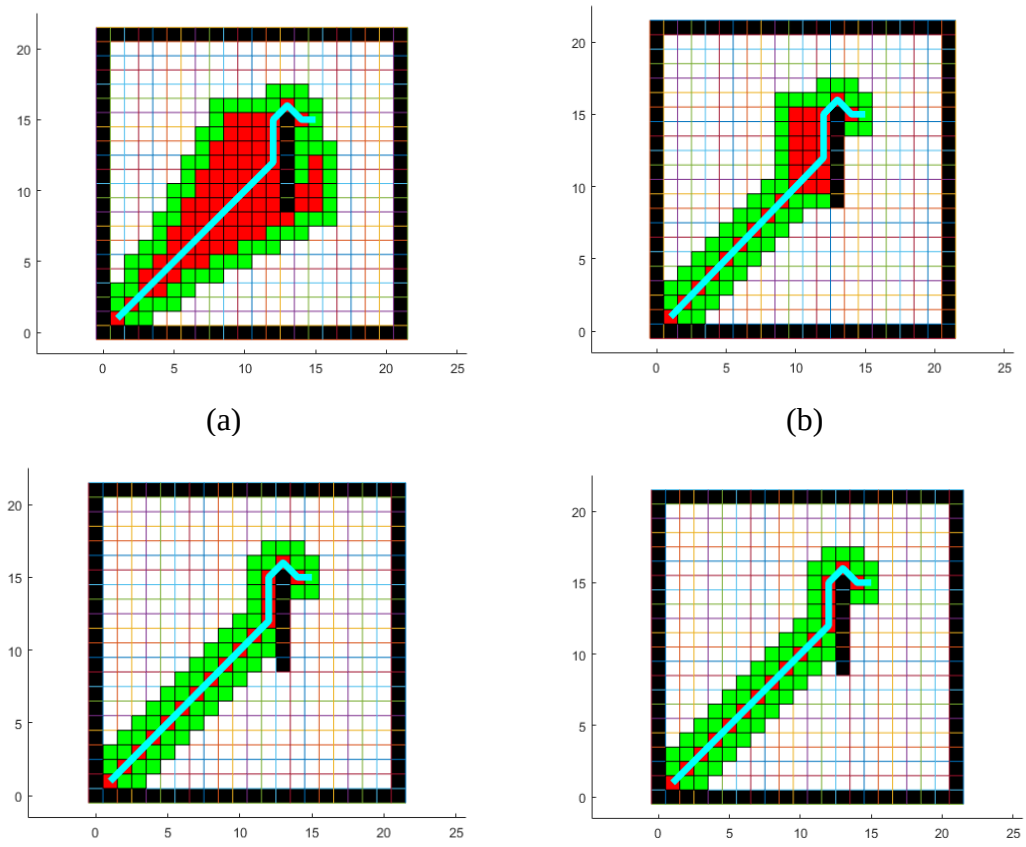

Fig 3. Weighted A * algorithm running diagram.
(a) W = 0.8; (b) w=1; (c) W=2; (d) W=20. (Photo/Picture credit: Original).

At w=0.8, 1, 2, 20, the following running data was obtained by executing the weighted A* Matlab code, as indicated in Table 2.

Table 2. The running result data of the weight A* algorithm under different weights (Group 1).

Value of weight	Path length	Computing time
0.8	104.5685	34.12
1	104.5685	4.32
2	104.5685	2.94
20	112.5685	2.03

As a control group, a second experiment is similarly set up. This study changed the path length and ran the MATLAB code again, obtaining the parameter that is displayed in Table 3.

Table 3. The running result data of the weight A* algorithm under different weights (Group 2).

Value of weight	Path length	Computing time
0.8	88.9706	23.12
1	88.9706	3.12
2	88.9706	2.14
20	112.5685	2.03

The information in Table 3 shows that, up to a certain distance, adding weight shortens the code execution time; however, this relationship is not evident between unnecessarily high weights; reduce the distance and the code execution time will go down below a specific weight.

The A* algorithm transforms into Dijkstra's algorithm, which ensures finding the shortest path when h (n) equals 0. The more nodes A* can expand and run more slowly if h (n) is less than or equal to g (n). Finding the shortest path can therefore be guaranteed, but the process is faster; nonetheless, when h (n) equals g (n), finding the shortest path cannot be guaranteed by merely looking for the optimal path without extending any additional nodes. Nevertheless, the process is quite quick, and the outcome ensures locating the shortest path; Finding the best path without extending any additional nodes cannot ensure finding the shortest path when h(n) is bigger than g(n), but the computation is faster; The A* algorithm becomes the BFS algorithm when h(n) is much larger than g(n). This algorithm is very fast in certain situations but cannot always find the shortest path; in large-scale state Spaces, for example, making the right decision can result in a relatively short path in a reasonable amount of time.

3.4. Weighted A* algorithm

An adaptation of the A* algorithm, the Weighted A* algorithm modifies search behavior by adding weights to the heuristic function h (n). These are a few of its key characteristics:

Adjustable heuristic function: The algorithm known as Weighted A* modifies the impact of the heuristic function h (n) by incorporating weight variables, commonly denoted by w. The weight factor w, whose value typically ranges from 0 to 1, can be utilized to strike a compromise between the path's cost (g(n)) and the significance of heuristic estimation h(n).

Avoid local minima: In some cases, the standard an algorithm may get stuck in local minima. By adjusting the weight factors, the Weighted An algorithm can reduce this risk as it allows the algorithm to explore other possible paths even when heuristic estimates are not entirely accurate.

Widely applicable: The Weighted A* algorithm can be applied to various path search and graph traversal problems, particularly in areas like intelligent cars and robot path planning.

3.5. D * Algorithm

The D* algorithm updates the cost of the node when the obstacle changes, and has the function of rejoining the node from the CloseList to the OpenList. This allows D* to adaptively meet changes in the path in A dynamic environment, whereas in A* algorithms, the entire path often needs to be recalculated if the environment changes.

Heuristic function: To determine the projected cost of traveling from the present node to the target node, D* employs the same heuristic function as A*, which is typically a cost or distance function.

Dynamic update: In contrast to A*, the D* method updates the node's state (i.e., known, unknown, or barriers) to control the search process. When the environment changes, only those nodes that are affected (such as new obstacles) need to be reassessed to improve efficiency.

Reverse search: During the update process, D* often performs a reverse search first, going backward from the target node to the starting point until an efficient and ideal path is discovered. This process is more efficient than a forward search that starts at the starting point, especially if the map changes less.

Memory Efficiency: The D * algorithm uses less memory since it only needs to keep the optimal path information from the starting point to the current node, as opposed to storing all potential paths across the search space.

Scalability: The D algorithm can be combined with other algorithms, such as the Liveness algorithm, to handle uncertainty in dynamic environments.

Local Consistency: The D * algorithm ensures the consistency of local paths, that is, when the environment changes, only the local paths related to the change will be recalculated.

4. Conclusion

This article is based on Matlab and deeply studies the A* algorithm. This article mostly discusses the dynamic A* algorithm and the weighted A* method. Firstly, the traditional A* algorithm was reviewed, and Dijkstra's algorithm and A* algorithm were compared, and a chart was drawn to identify the advantages and disadvantages of the traditional algorithm. Then, the Weighted A* algorithm was further studied. By applying weight w, the heuristic function was optimized and improved, and qualitative and quantitative studies were conducted by running Matlab code. For qualitative research, the equivalent case of the weight A* algorithm was studied when w=0, w=1, and w approaches infinity. For quantitative research, further investigation was conducted on the differences in running time under different weights and paths. A chart was drawn and the data was analyzed, and the conclusion was drawn that the larger the value of w, the more effective it is in reducing algorithm running time. Lastly, the A* algorithm variation known as the dynamic A* algorithm was examined. It is determined that the dynamic A* algorithm contains features like reverse exploration and real-time updates by utilizing Matlab to gather pertinent information about it; The weighted A* algorithm and dynamic A* algorithm studied in this article are further optimization and improvement of the algorithm, which is of great significance for exploratory problems such as mazes and expands the research direction of the algorithm. However, this study focuses on the research of the algorithm itself and lacks experimental verification. In the future, this research can be combined with various algorithms such as neural networks and large-scale models, further combined with directions such as artificial intelligence, big data, and mechanical algorithms, and applied to industries such as manufacturing and even agriculture.

References

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

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

[3] Shi Y., Yang J., Bu S., Zhu L. Intelligent vehicle path planning algorithm based on improved RRT. Computing Technology and Automation, 2019, 38 (04): 81-86.

[4] Sariff, N., & Buniyamin, N. An overview of autonomous mobile robot path planning algorithms. In 2006 4th student conference on research and development. IEEE, 2006, pp. 183-188.

[5] Bortoff, S. A. Path planning for UAVs. In Proceedings of the 2000 American Control Conference. IEEE. 2000, 1(6), pp. 364-368.

[6] Wang, C., Wang, L., Qin, J., Wu, Z., Duan, L., Li, Z., et al. Path planning of automated guided vehicles based on improved A-Star algorithm. In 2015 IEEE International Conference on Information and Automation. IEEE, 2015, pp. 2071-2076.

[7] Tang, G., Tang, C., Claramunt, C., Hu, X., & Zhou, P. Geometric A-star algorithm: An improved A-star algorithm for AGV path planning in a port environment. IEEE Access, 2021, 9, 59196-59210.

[8] Erke, S., Bin, D., Yiming, N., Qi, Z., Liang, X., & Dawei, Z. An improved A-Star based path planning algorithm for autonomous land vehicles. International Journal of Advanced Robotic Systems, 2020, 17(5), 1729881420962263.

[9] Candra, A., Budiman, M. A., & Pohan, R. I. Application of a-star algorithm on pathfinding game. In Journal of Physics: Conference Series. 2021, 1898(1), p. 012047.

[10] XiangRong, T., Yukun, Z., & XinXin, J. Improved A-star algorithm for robot path planning in static environment. In Journal of Physics: Conference Series. 2021, 1792(1), p. 012067.

[11] Liu, L., Wang, B., & Xu, H. Research on path-planning algorithm integrating optimization A-star algorithm and artificial potential field method. Electronics, 2022, 11(22), 3660.

[12] Cui, S. G., Wang, H., Yang, L., A Simulation Study of A-star Algorithm for Robot Path Planning. 16th international conference on mechatronics technology, 2012, pp. 506-510.

[13] Duchoň, F., Huňady, D., Dekan, M., & Babinec, A. Optimal navigation for mobile robot in known environment. Applied Mechanics and Materials, 2013, 282, 33-38.