Fast Jump Point Search Based Path Planning for Mobile Robots

Yucong Tong, Huaiyu Wu, Xiujuan Zheng, Zhihuan Chen and Yang Chen

March 12, 2021

Fast Jump Point Search Based Path Planning for Mobile Robots

1st *Yucong Tong*
Institute of Robotics and Intelligent Systems, Wuhan University of Science and Technology
e-mail:573817112@qq.com

2nd *Huaiyu Wu*
Institute of Robotics and Intelligent Systems, Wuhan University of Science and Technology
Engineering Research Center for Metallurgical Automation and Measurement Technology of Ministry of Education
e-mail: wuhy@wust.edu.cn

3rd *Xiujuan Zheng*
Institute of Robotics and Intelligent Systems, Wuhan University of Science and Technology
Engineering Research Center for Metallurgical Automation and Measurement Technology of Ministry of Education
e-mail: zxj@wust.edu.cn

4th *Zhihuan Chen*
Institute of Robotics and Intelligent Systems, Wuhan University of Science and Technology
Engineering Research Center for Metallurgical Automation and Measurement Technology of Ministry of Education
e-mail: czh@wust.edu.cn

5th *Yang Chen*
Institute of Robotics and Intelligent Systems, Wuhan University of Science and Technology
Engineering Research Center for Metallurgical Automation and Measurement Technology of Ministry of Education
e-mail: chenyag@wust.edu.cn

Abstract: We propose a method to improve the performance of JPS for path planning on static grid map, including a fast neighbor pruning method and a symmetry-breaking heuristic function. First, we adopt a new and effective method to identify jump points quickly through bit operations at a single time. Second, we adopt a symmetry-breaking heuristic function to pruning redundant jump points by adding angle information and node sequence number in order to further speed up path planning search. We conducted simulation experiments on grid maps of different specifications and obstacle ratios to verify the effectiveness and feasibility of the proposed algorithm, comparing with JPS. The experimental results show that our improvement has more advantages on grid maps of different specifications and obstacle ratios, both in terms of search time and number of jump points.

Keywords—path planning, prune rules, jump point search, heuristic function, grid map

I. Introduction

With the rise of artificial intelligence, path planning algorithms are widely used in fields as robotics, video games, autonomous driving, and map navigation. The purpose of path planning is to specify a collision-free and safe path from the start point to the target point for the moving object. The difficulty of path planning is how to efficiently realize the optimal path in a specific environment. The Dijkstra algorithm and A* algorithm [1] based on grid search are classic methods to solve the static path planning of mobile robots. However, these algorithms are difficult to perform well when processing large amount of data. Therefore, search speed and memory overhead already become the bottleneck of this type of algorithm. In 2011, Harabor D D [2] proposed JPS(Jump Point Search) which was considered to be the fastest static grid search algorithm at the time. JPS selectively expands nodes through a set of simple rules, while ensuring the optimal path, greatly reduce the amount of calculation for searching nodes.

Although many achievements have been made in the research of JPS algorithm and its variants, there are still many challenges in terms of path optimization, time cost and memory consumption. Xiaolu Ma [3] applied the bidirectional search to JPS to improve the overall search efficiency and solve the problem of insufficient real-time performance in mobile robot navigation; Anti Li [4] used JPS as the global path planning in UAV navigation, it guided the subsequent obstacle avoidance process and shorten the task time. Kaijun Zhou [5] modified JPS framework as a local obstacle avoidance module for unmanned driving, and verified the feasibility of JPS in local path planning. Traish J [6] proposed Boundary Lookup JPS used preprocessing technology to mark the map boundaries in the grid, reduce the overall number of natural neighbors. Hu Y [7] continued their previous work by pre-storing jump points to speed up the jump point search process and improved the path search efficiency. Algfoor Z A [8] improved the evaluation function of JPS

This work is supported by National Natural Science Foundation of China under Grant No.61573263.

and proposed three heuristic functions with weighting factors to optimize the search space and memory consumption. These above methods have improved JPS from natural neighbors, jump points and heuristic functions, but how to further trim the symmetry nodes in the grid, reduce the number of jump points, and improve the efficiency of the algorithm still a problem worthy of research.

It is found that the key to improve JPS lies in the identification of jump points and the acceleration of the iterative process. We improved the neighbor pruning rules and heuristic functions, and proposed a Fast Jump Point Search (Fast JPS) algorithm. We adopted a fast prune rule by analyzing the characteristics of the grid and introducing bit operations, the search for the current node is converted into a batch search for obstacles, and natural neighbors and forced neighbors are directly extracted based on the obstacle location information. At the same time, combined with the heuristic function that destroys symmetry, the symmetry nodes in the grid are selected to trim redundant nodes and speeding up the search for jump points. The simulation results show that our improved algorithm have a better path planning performance than JPS on grid maps of different specifications and obstacle ratios.

II. JPS ALGORITHM IS USED FOR PATH PLANNING

A. *Overview of the principle of JPS algorithm*

The main idea of JPS is to skip a large number of symmetrical path nodes in the search process, and only look for the jump points as the nodes to be expanded. So as to quickly search out the path connecting the start point and the target point. In essence, JPS is an optimization algorithm of A*. In each node search, JPS starts to move along the straight or diagonal direction of the current node, and prunes the symmetric nodes with long paths in the neighbors according to certain rules. Only search for nodes with the shortest path among neighbors, call them natural neighbors or forced neighbors, and then further figure out nodes with expansion value, call them jump points, and add them to the *openlist* to complete the heuristic search recursively Process [9]. The core of JPS can be summarized into two rules: neighbor pruning rules and jump point screening rules.

B. *Pruning rules of neighbor*

The neighbor pruning rule only trims the eight neighbor nodes of the current node each time, and distinguishes the trimmed neighbors from the preserved natural neighbors based on the path length. According to the moving direction and obstacles, there are four situations, as shown in Figure 1.

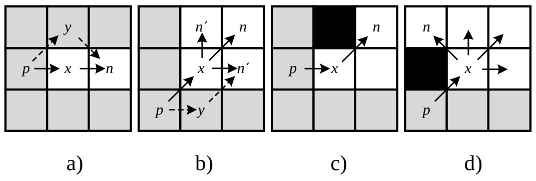


Fig. 1. When the move from *p* to *x* is straight (as in a)) only one natural neighbor remains. When the move from *p* to *x* is diagonal (as in b)), three natural neighbors remain. When obstacles are adjacent to *x* some neighbors become forced; we illustrate this in for straight moves (as in b)) and for diagonal moves (as in d)).

In Figure 1, x is the current node, p is the parent node, n is the target node, $x = < p, x, n >$ represents the path from p to n through, as shown by the solid line, $\pi' = < p, y, n >$ represents any path from p to n without x passing through, as shown by the dashed line, $len(\cdot)$ represents the path length.

1) *There are no obstacles*

As shown in Figure 1a) straight movement, the length of π the path is shorter than the length of π' the path, indicating that the neighbor n is the shortest path node in the current neighbor, the white node is the natural neighbor, and the gray nodes that are symmetrical on both sides of the straight line will be trimmed. The pruning rules of natural neighbors are expressed as:

$$len(< p, x, n >) < len(< p, y, n >) \qquad (1)$$

In Figure 1b) diagonal movement, there is a path π' with the same length as the path π. At this time, in addition to neighbors n, neighbors n' are also natural neighbors. The pruning rules of natural neighbors becomes:

$$len(< p, x, n >) \le len(< p, y, n >) \qquad (2)$$

2) *There are obstacles*

As shown in Figure 1c) and Figure 1d), the path to the neighbor n changes due to the presence of obstacles. Obviously, the path π is the shortest path node at this time. Therefore, in addition to the natural neighbor in the absence of obstacles, an additional natural neighbor needs to be added, which is called a forced neighbor. The forced neighbor pruning condition is:

$$len(< p, x, n >) < len(< p, y, n >) \qquad (3)$$

C. *Pruning rules of jump point*

The neighbor pruning rule preserves the shortest path node in the neighbors. However, not all natural neighbors are the final path nodes. For the target path, only nodes that change the direction of the path have the value of expansion. The purpose of trimming jump points is to replace the middle jump points with no expansion value by the remote jump points. The pruning rules of jump point as follows:

1) When the natural neighbor is the start point or the target point, it must be a jump point;

2) The natural neighbor has at least one forced neighbor. At this time, the natural neighbor and the forced neighbor are jump points for each other;

3) When searching in the diagonal direction, if there is a jump point in the horizontal or vertical direction of the natural neighbor, the natural neighbor on the diagonal is also a jump point.

D. Description of JPS path planning problem

In the robot path planning problem, the implementation framework of JPS and A* are similar. In order to apply JPS to actual robots, the actual map should be converted into a binary grid map, including the feasible area and the obstacle area. Irregular obstacles are filled according to their grid coordinates, and the robot can be regarded as a mass point. The JPS path planning process is shown in Figure 2.

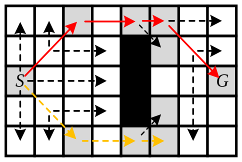


Fig. 2. JPS path palnning process

As shown in Figure 2. At the start point, JPS moves along the straight and diagonal directions, searching for only one node at a time. According to pruning rules of neighbor and jump point, the neighbors of the current node are divided into natural neighbors and forced neighbors. White nodes are pruned neighbors and gray nodes are jump point. Each time the search stops at the point where the jump point is trimmed or the boundary is reached, the jump point returned will be add to *openlist*; then, select the optimal jump point from the heuristic function, and delete the jump point from *openlist* and add to *closelist* ; Repeat the above process until the target point is found. The final path consists of the parent nodes backtracked in *closelist*, as shown by the solid red line. From the above process, it can be seen that JPS has two shortcomings in path planning:

1) Each pruning is only for the neighbors of the current node, and the efficiency of searching paths in large scenes needs to be improved;

2) When facing a symmetrical obstacle as shown in Figure 2, JPS will search for all the symmetry jump points on the red and yellow paths. And the heuristic function estimates of the points are the same, which causes the calculation amount to increase exponentially.

III. Fast JPS Algorithm

In response to the above problems, we propose a Fast JPS algorithm to improve the efficiency of node pruning and reduce the search for symmetry paths. The algorithm consists of two parts: accelerating the pruning of nodes and the identification of jump points with bit operations; and using symmetry-breaking heuristic function replace the conventional heuristic function to reduce the number of jump points.

A. fast pruning rules

JPS only trims the neighbors of the current node, and selects jump points one by one; Hu Y [7] use bit operations to implement prune of neighbors and jump points; we improved bit operations according to the location of obstacles. The information is directly extracted to force neighbors, and the functions of quickly pruning nodes and screening jump points are realized to improve the overall performance of the algorithm. We take Figure 3 as an example to illustrate the whole process

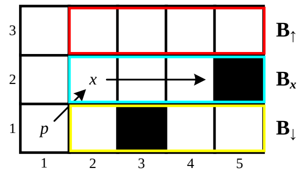


Fig. 3. Schematic of fast pruning rules

In Figure 3 moving along the diagonal form p to x. According to JPS flow, the first step is to move horizontally to the right in a straight line at $<2,2>$. Pruning identify jump point $<3,2>$ and forced neighbor $<4,1>$ and stop before $<5,2>$. As mentioned in c) of pruning rules of jump point, x is also a jump point and there are no return in vertical direction. So far, the pruning of $<2,2>$ ends. The above jump points will be arranged by the heuristic function, and the optimal value will be used as the current node for the next expansion. If x is not a jump point, move one step diagonally to $<3,3>$ and repeat the above process.

Hu Y [7] uses bit operations to implement batch pruning of nodes. The implementation process is as follows: first, detect the dead end of the current node's moving direction; second, detect the forced neighbors of adjacent rows of the current node; finally, use logic “or” operation to detect jump points.

We propose a new pruning rules based on bit operation method that can speed up the identification of jump points without violating pruning rules of neighbor and pruning rules of jump point. It can be seen from Figure 1 that forced neighbors only exist near obstacles. At the same time, only natural neighbors that force neighbors are jump points. In other words, we only need to find the node near the obstacle and judge whether it is a jump point.

As shown in blue, red and yellow boxes in Figure 3, the current row of x is B_x, and the up row of x is $B_\uparrow$, and the down row of x is $B_\downarrow$. The three-row grid box can be expressed as:

$$\begin{cases} B_\uparrow = [0,0,0,0] \\ B_x = [0,0,0,1] \\ B_\downarrow = [0,1,0,0] \end{cases} \tag{4}$$

Equation (4), 0 means free and 1 means obstacle. Perform a series of operations as follow:

1) Detect forced neighbor: if there is a potential forced neighbor among the neighbors of x, it means:

$$B_\uparrow(i:i+1) \mid B_\downarrow(i:i+1) = [1,0] \tag{5}$$

2) Detect jump point: if the nodes are free on the moving direction of x, x is a jump point. It means:

$$B_x(i:i+1) = [0,0] \tag{6}$$

3) Detect stop: if (6) is not established, it means that there is a dead end and stop; if (6) is established and (5) is unestablished, it means that there is no jump point, $i >> 1$.

It can be judged that $B_x(i=2) =< 3,2 >$ is a jump point based on the above conditions. The result is consistent with JPS. But the fast pruning rules based on bit operation can complete the pruning of nodes and the identification of jump points at one time.

B. Heuristic functions that break symmetry

Qiu L [1] combined JPS with A* to figure out path planning, and used Euclidean distance to calculate heuristic function. On a grid map that allows straight and diagonal movement, the Euclidean distance is usually shorter than the actual path distance, causes the path length to be sub-optimal. In order to improve the accuracy of path estimation, we choose *Octile* distance allowed to move diagonally as the heuristic function. The calculation method of *Octile* is:

$$Octile = dx + dy + (\sqrt{2} - 2) * \min(dx, dy) \tag{7}$$

Equation (7), dx and dy represents the distance that the current node is projected from the target point on the coordinate system. At the same time, in order to make the moving direction close to the target point to accelerate the path convergence, we add the angle information to the heuristic function:

$$h(n) = \alpha + \cos\theta \tag{8}$$

Equation (8) θ is the angle made up of the line from the start point to the target point and the line from the current node to the target point. θ can be calculated by the rotation angle formula. Some nodes can be trimmed by (8).

But when the obstacle is symmetrical as shown in Figure 4, JPS cannot distinguish the symmetry nodes resulting in repeated searches.

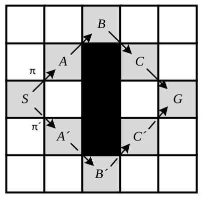


Fig. 4. Symmetry search

In Figure 4, because of the symmetrical nodes AA', BB', CC', there are two similar paths: $\pi =< A, B, C >$ and $\pi' =< A', B', C' >$. In order to break the balance of the symmetrical nodes and reduce invalid searches and nodes, we adopt the heuristic function as follow:

$$h(n) = (1 + p) * (\alpha + \cos\theta) \tag{9}$$

The function of (9) is to distinguish the nodes with the same value of the heuristic function. The form of p is the key to the heuristic function. It can be designed as:

$$p = \frac{index * l}{\sum_{i=0}^{N} l_i} \tag{10}$$

I*ndex* indicate the node serial number, l indicate the minimum distance of unit grid and $\sum_{i=0}^{N} l_i$ indicate the longest path distance in the current grid map.

IV. Simulation Experiment and Result Analysis

In order to verify the feasibility and effectiveness of the Fast JPS we proposed, we implement JPS and Fast JPS under the grid map of different scenes. The main performance indicators of the test include: number of jump points, path length, search time. The computer configuration is

Intel Corei5 4G, win10 system, and the simulation environment is MATLAB2015a. The simulation includes three scenarios. In the grid map, the black grid represents the obstacle area, the gray grid represents the start point S or the target point G, the white gird represents the feasible area, and the red solid line is the result of path planning. We stipulate that the path can move along straight or diagonal, and cannot touch obstacles. The blue point indicates that the grid is a jump point. The result of path planning as follow:

Scene 1 in Figure 5 is a typical symmetry structure, which is used to verify the screening effect of the symmetry-breaking heuristic function we designed;

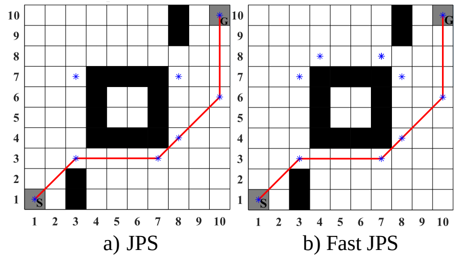


Fig. 5. Test results of the two algorithms in scene 1

Scene 2 in Figure 6 contain two paths with the same length and the same number of jump point, which is mainly used to verify the pruning rules we have improved;

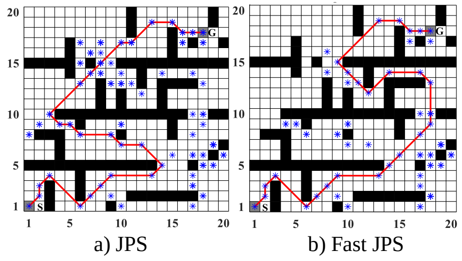


Fig. 6. Test results of the two algorithms in scene 2

Scene 3 in Figure 7 is a large-scale grid with many invalid nodes, which is used to verify the grid preprocessing technology.

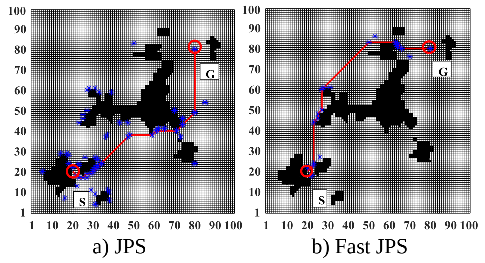


Fig. 7. Test results of the two algorithms in scene 3

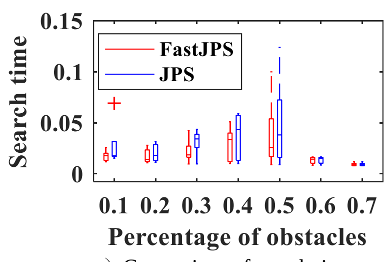
We conduct further analysis on the statistics of the data results of the three scenes in Table 1.

TABLE I. PERFORMANCE TEST RESULTS OF JPS ALGORITHM AND FAST JPS ALGORITHM IN THREE SCENARIOS

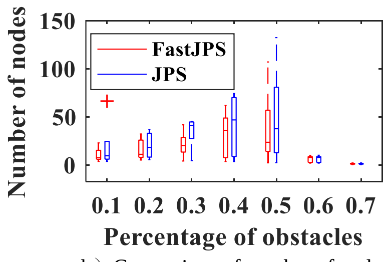
	Performance	**JPS**	**Fast JPS**	**Improve**
Scene 1	**Number of jump point**	10	7	30%
	Search time	9.22ms	8.31ms	9.9%
	Path length	15.07	15.07	-
Scene 2	**Number of jump point**	75	62	17.3%
	Search time	29.21ms	25.95ms	11.1%
	Path length	49.36	49.36	-
Scene 3	**Number of jump point**	89	63	29.2%
	Search time	105.14ms	64.80ms	38.3%
	Path length	103.67	103.67	-

The results of the test in the above scene as show in the Table 1. It can be seen that our method is better than JPS in terms of overall performance. In three different scenarios, our method has a certain improvement in reducing the number of jump points and searching time. Even in the symmetrical gird map and large-scale gird map, our method is still much better.

To verify the effectiveness of Fast JPS in irregular grid maps. We conducted experiments on 100×100 grid maps. Obstacles on each map are randomly generated, and their ratio is 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7. 100 iterations for each map. We separately counted the search time and the number of jump points for valid paths, and eliminated the data that could not generate paths. We show the results of the experiment in box plot in Figure 8.



a）Comparison of search time



b）Comparison of number of nodes

Fig. 8. Box plot of two algorithms on 100 × 100 random grid

As shown in Figure 8, we have summarized the mean and variance of the time and jump points of the two algorithms under a 100×100 grid. As the proportion of obstacles increases, the number of jump points and search time trends extended by the two algorithms are consistent. When the obstacle ratio is 0.1, JPS has a larger outlier, and our method is relatively stable. Table 2 shows the statistical results of a 100×100 random grid, including the minimum Min, 25% quartile Q1, median Mid, 75% quartile Q3, and maximum Max. It can be seen from the table that our method has a certain improvement in the total number of jump points and search time in the entire data segment. At the same time, the outliers have been reduced. The results prove that our proposed method has good stability and feasibility.

TABLE II. STATISTICS RESULTS OF 100×100 RANDOM GRID

Percentage of obstacle		**0.1**	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**
Number of nodes								
Min	**JPS**	3.7	5.4	4.4	3.5	2.3	1.9	1.2
	Fast JPS	3.7	5.1	4.1	3.6	2.2	1.9	1.2
Q1	**JPS**	6.8	9.1	44.5	10.4	16.1	7.8	0.3
	Fast JPS	6.6	8.8	41.7	8.8	17.9	7.7	0.3
Mid	**JPS**	9.5	18.2	40.8	46.8	37.6	10.1	1.5
	Fast JPS	7.4	11.1	20.1	35.6	23.6	10	1.5
Q3	**JPS**	10.6	31.7	45.2	74.2	63.6	2.9	1
	Fast JPS	12.7	23.7	24.1	44.4	40.2	2.9	1
Max	**JPS**	66.3	36.7	35	68.6	132.3	8	2.1
	Fast JPS	22.9	32.2	16.5	61.8	107	8	2.1
Search time(ms)								
Mid	**JPS**	35.2	22.6	19.3	19.7	17.7	17.1	15.7
	Fast JPS	35.4	21.5	19.2	20.0	16.7	16.8	15.2
Q1	**JPS**	30.6	26.9	87.3	28.6	36.9	30.5	15.8
	Fast JPS	23.4	24.9	85.2	25.1	39.0	28.6	15.7
Mid	**JPS**	34.9	36.1	68.3	86.7	76.1	31.5	17.9
	Fast JPS	26.9	27.8	37.0	66.9	51.5	31.2	18.0
Q3	**JPS**	38.6	54.9	74.2	117.8	110.4	22.0	19.3
	Fast JPS	36.6	43.7	43.9	72.4	76.9	21.9	19.4
Max	**JPS**	138.1	63.4	62.0	113.3	247.4	33.1	23.9
	Fast JPS	51.5	55.9	36.9	101.8	199.7	33.0	23.7

V. SUMMARY

Aiming at the pruning rules and symmetry jump points in JPS, we propose improvements from two aspects. First, on the basis of analyzing the pruning rules and jumping point rules, we use bit operation and obstacle information to extract the jumping points. This method can effectively simplify the intermediate process. Second, for the excessive calculation caused by the symmetrical jump points. We propose a heuristic function for symmetry destruction. We add a controllable amount to the result of the heuristic function to reduce the generation of symmetry jump points. The simulation results show that our method generates fewer jump points and less search time.

References

[1] Qiu L. “Speed-up of A* pathfinding with jump point search algorithm,” J. Journal of Lanzhou University of Technology, 2015,41(3):102-107.

[2] Harabor D D, Grastien A. “Online Graph Pruning for Pathfinding on Grid Maps,”C. Aaai Conference on Artificial Intelligence. DBLP, 2011:1114–1119.

[3] Xiaolu Ma, Hong Mei. “Research on global path planning of mobile robot based on bidirectional hop search algorithm,” J. Mechanical science and technology, 2020, 39 (10): 1624-1631. 马小陆，梅宏。双向跳点搜索算法的移动机器人全局路径规划研究［J］。机械科学与技术，2020，39（10）：1624－1631。

[4] Anti Li, Chenglong Li, Dingjie Wu, Peng Wei. “Collision avoidance decision method of UAV random search combined with jump point guidance.” Acta Aeronautica Sinica, 2020, 41 (08): 325-337. 李安醍，李诚龙，武丁杰，卫鹏。结合跳点引导的无人机随机搜索避撞决策方法［J］。航空学报，2020，41(08)：325-337。

[5] Kaijun Zhou, Lingli Yu, Ziwei Long, Siyao Mo. “Local Planning of Driverless Car Navigation Based on Jump Point Search Method Under Urban Environment,” Future Internet, 2017, 9(3):459-468.

[6] Traish J, Tulip J, Moore W. “Optimization using boundary lookup jump point search,” J. IEEE Transactions on Computational Intelligence & Ai in Games, 2017, 8(3):268-277.

[7] Hu Y, Harabor D, Qin L, et al. “Regarding Goal Bounding and Jump Point Search,” J. Journal of Artificial Intelligence Research, 2021, 70:631-681.

[8] Algfoor Z A, Sunar M S, Abdullah A. “A new weighted pathfinding algorithms to reduce the search time on grid maps,” J. Expert Systems with Applications, 2017, 71(APR.):319-331.

[9] Min J G, Ruy W S, Park C S. “Faster pipe auto-routing using improved jump point search.” International Journal of Naval Architecture and Ocean Engineering, 2020, 12:596-604.

[10] Ma L, Gao X, Fu Y, et al. “An Improved Jump Point Search Algorithm for Home Service Robot Path Planning,” 2019 Chinese Control And Decision Conference (CCDC). IEEE, 2019(2):1162-1167.

[11] V V Maneev, M V Syryamkin. “Optimizing the A* search algorithm for mobile robotic devices,” J. IOP Conference Series: Materials Science and Engineering, 2019, 516(1):012054.

[12] Harri A. “Using the Hierarchical Pathfinding A* Algorithm in GIS to Find Paths through Rasters with Nonuniform Traversal Cost,” J. International Journal of Geo-Information, 2013, 2(4):996-1014.

[13] Bagli S, Geneletti D, Orsi F. “Routeing of power lines through least-cost path analysis and multicriteria evaluation to minimise environmental impacts.” Environmental Impact Assessment Review, 2011, 31(3):234-239.

[14] Daniel Harabor,Alban Grastien. “Improving Jump Point Search,”[C]// Proceedings of the twenty-fourth international conference on automated planning and scheduling: ICAPS 2014, Portsmouth, United Kingdom, 21-26 June 2014.2014:128-135.