A jump point search improved ant colony hybrid optimization algorithm for path planning of mobile robot

Tao Chen1,2, Suifan Chen1, Kuoran Zhang1, Guoting Qiu2, Qipeng Li1 and Xinmin Chen2

Abstract
To improve the finding path accuracy of the ant colony algorithm and reduce the number of turns, a jump point search improved ant colony optimization hybrid algorithm has been proposed in this article. Firstly, the initial pheromone concentration distribution gets from the jump points has been introduced to guide the algorithm in finding the way, thus accelerating the early iteration speed. The turning cost factor in the heuristic function has been designed to improve the smoothness of the path. Finally, the adaptive reward and punishment factor, and the Max–Min ant system have been introduced to improve the iterative speed and global search ability of the algorithm. Simulation through three maps of different scales is carried out. Furthermore, the results prove that the jump point search improved ant colony optimization hybrid algorithm performs effectively in finding path accuracy and reducing the number of turns.

Keywords
Mobile robot, path planning, jump point search algorithm, ant colony optimization algorithm, turn cost factor

Introduction
As an important part of kinematics, path planning is widely used in unmanned aerial vehicle (UAV),1 robot,2 and automated guided vehicle (AGV).3 Path planning for mobile robots refers to planning a path from starting node to the target node. It requires the path to be safe, collision-free, and the comprehensive indicators such as distance, time, and energy consumption are optimal.4

Currently, a lot of scholars have done much research on the path planning problem and proposed a wide variety of algorithms, such as A* algorithm,5 ant colony optimization (ACO) algorithm, genetic algorithm (GA), particle swarm optimization algorithm,6 artificial potential field method (APF), fireworks algorithm (FA), and so on.

ACO algorithm is a heuristic algorithm first proposed by Dorigo and Maniezzo in the 1990s,11 and it imitates the foraging behavior of ant colonies in nature. It’s widely used in path planning because of its strong robustness, positive feedback enhancement of optimal path, and ease of...
combination with other algorithms. However, there are some problems, such as slow convergence speed, easy trap into a locally optimal solution. To address these problems, plenty of scholars have conducted in-depth research. Sun et al.\textsuperscript{12} greatly improved the survival number of ants in the algorithm and accelerated the convergence rate of the algorithm by adopting different strategies for deadlock problems in the algorithm. Zhang et al.\textsuperscript{13} improved the ability of search space by using information entropy to describe the ant colony and construct a nonuniform distribution of pheromone concentration and renewal mechanism. Zhang et al.\textsuperscript{14} accelerated the convergence rate of the algorithm and operating efficiency by dividing the ant colony into guide layer ant colony and common layer ant colony to search path. Khaled and Farid\textsuperscript{15} accelerated the algorithm’s convergence rate and increased the search scope by using a new pheromone updating rule and dynamic evaporation rate. Luo et al.\textsuperscript{16} made the initial pheromone uneven distribution to avoid the early blind search of the algorithm, speed up the convergence of the algorithm, and used the dynamic punishment method to reduce the number of ants trapped in a deadlock. Meng et al.\textsuperscript{17} proposed a multi-ant colony cooperative optimization algorithm based on a cooperative game mechanism to accelerate the convergence speed of the algorithm. Tao et al.\textsuperscript{18} used fuzzy control to change the value of the pheromone heuristic factor and expectation heuristic factor and adjusted the evaporation rate in stages, which ensured the global search ability of the algorithm.

With the development of algorithms, the ACO algorithm gradually began to combine with other algorithms to achieve the purpose of an optimization algorithm. Ma et al.\textsuperscript{19} used the improved FA to generate the initial pheromone concentration values to distribute them on the map and then search for a globally optimal path through the ACO algorithm. This algorithm can find the optimal path in the case of fewer iterations. Dai et al.\textsuperscript{20} accelerated the algorithm’s convergence rate and improved the path’s smoothness by adding evaluation functional and bending inhibitors of the A* algorithm. Zhu et al.\textsuperscript{21} improved the fitness function of the ant colony algorithm by using the APF algorithm, and the adjustment of negative feedback and self-fitness function was taken into account at the same time. Therefore, this method can greatly accelerate the convergence rate of the algorithm. Yu\textsuperscript{22} gave the concept of an ant colony particle and fused the particle swarm algorithm with the ACO algorithm to improve the convergence speed of the algorithm. Lei et al.\textsuperscript{23} used the GA to generate the initial path and took the better path as the reference information of the ant colony algorithm pheromone initialization to reduce the blindness of the initial search of the ant colony algorithm. Then, the ant colony algorithm was used to further optimize the initial path and accelerate the convergence speed of the algorithm. Saenphon et al.\textsuperscript{24} used the fast reverse gradient method to obtain the possible optimal solution and then used the ACO algorithm to further improve the quality of the solution. Li et al.\textsuperscript{25} proposed optimization ants and reconnaissance ants combined with the artificial bee colony algorithm and assigned different pheromone updating weights, respectively. The convergence speed of the algorithm is improved to avoid the algorithm falling into local optimum and ensure the diversity of solutions.

Based on the above literature research, the JPIACO algorithm is proposed in this article. The main contributions of this work are as follows:

1. The initial pheromone concentration distribution obtained from the jump point search (JPS) algorithm is designed to accelerate the early iteration speed.
2. A new heuristic function is designed to improve the smoothness of the path.
3. A new adaptive reward and punishment factor is designed to improve the iterative speed.
4. The Max–Min ant system has been introduced to improve the global search ability.

The framework of the article is as follows. The second section describes the map establishment method. The third section mainly describes the JPS algorithm. The fourth section mainly describes the JPIACO algorithm. The fifth section describes the experiments and results. The sixth section summarizes the thesis.

**Environment model**

In the path planning of mobile robots, we should establish a 2D model of the robot movement environment firstly, and the modeling methods include the free space method, grid method, topology method, and so on. Simple, effective, easy to implement, and can clearly express obstacles are considered, the grid method to model is used in this article. An example of the grid map is shown in Figure 1.

In the grid map, white grids are free spaces that robots can move and black grids are obstacle spaces that robots can’t...
move. To save the path and pheromone concentration easily, numbered the grid map by the way from left to right and top to bottom. As the numbers show in Figure 1, from left to right and top to bottom, the map numbers the grid starting at 1 until it is added to the 100th grid in the lower right corner.

The relationship between its coordinates and the number \( n \) of the grid is shown in equation (1)

\[
\begin{align*}
X_n &= \begin{cases} 
\text{mod}(n, MM) - 0.5, & X_n \neq -0.5 \\
MM - 0.5, & X_n = -0.5 
\end{cases} \\
Y_n &= \text{ceil} \left( \frac{n}{MM} \right) - 0.5
\end{align*}
\]

(1)

where \( MM \) is the number of grids per row, \( \text{mod} \) represents the operation of complementation, and \( \text{ceil}(n/MM) \) represents the integer operation not less than \( n/MM \).

**Initial path planning based on the JPS algorithm**

The JPS algorithm is the optimization of the A* algorithm. In the traditional A* algorithm, path analysis of each step needs to calculate the value of the eight direction nodes around, it generates a lot of unnecessary calculation steps which reduces the efficiency of the algorithm. While the JPS algorithm discards unnecessary nodes, leaving only representative nodes in the path search (these nodes are called jump points), it greatly improves the running efficiency of the algorithm.

**Jump point screening**

In the grid map, use each grid as a node, define eight grids adjacent to the current node grid as “around the current node,” the current node position is the parent node \( P(x) \) and the next node to move is \( x \). There are two cases in jump point screening, one is when there are no obstacles around the current node, and the other is when there are obstacles around the current node.\(^2\)

\( \text{Case 1: There are no obstacles around the current node} \)

In Figure 2(a), when searching from point \( P(x) \) to the right, the path generated by \( P(x) \) passing through node \( x \) to the next node \( n \) (such as \( 4 \rightarrow 5 \rightarrow 6 \)) in this direction is the shortest compared with \( P(x) \) to the next node \( n \) without through node \( x \) (such as \( 4\rightarrow \cdot \cdot \cdot \rightarrow 6 \) without through node 5), so the gray nodes (1, 2, 3, 4, 7, 8) are the unnecessary node and does not need to calculate the path generation value, define these gray nodes as invalid nodes. Therefore, when there are no obstacles around the node, the screening conditions in the horizontal and vertical directions are as follows

\[
L(P(x), \ldots, n/x) \geq L(P(x), x, n)
\]

(2)

where \( L(\cdot) \) represents path length, \( <P(x), \ldots, n/x> \) is a path from \( P(x) \) to \( n \) without going through \( x \); \( <P(x), x, n> \) is the path \( P(x) \rightarrow x \rightarrow n \) and \( n \) is the movable node around node \( x \).

In Figure 2(b), when searching from point \( P(x) \) to the diagonal direction, similarly, gray nodes (1, 2, 3, 4, 7) are invalid nodes and do not need to calculate the path generation value, define the white nodes around node \( x \) as natural neighbors. Therefore, when there are no obstacles around the node, the screening condition in the diagonal direction is as follows

\[
L(P(x), \ldots, n/x) > L(P(x), x, n)
\]

(3)

\( \text{Case 2: There are obstacles around the current node} \)

As shown in Figure 3, there is an obstacle node (2 in Figure 3(a) and 4 in Figure 3(b)) around node \( x \), the method of searching nodes with no obstacles cannot be used for screening, and a new screening rule must be adopted. In this time, the distance from \( P(x) \) to the next node \( n \) through node \( x \) (such as \( 4 \rightarrow 5 \rightarrow 3 \) in Figure 3(a)) is the shortest path compared with \( P(x) \) to the next node \( n \) without through node \( x \) (such as \( 4 \rightarrow \cdot \cdot \cdot \rightarrow 3 \) without through node 5 in Figure 3(a)), and node \( n \) is not a natural neighbor of node \( x \), call the node \( n \) is the mandatory neighbor of node \( x \). Therefore, MN in Figure 3(a) and 3(b) are mandatory neighbor points. The screening conditions for mandatory neighbor points are as follows:

\[ P(x) \rightarrow x \rightarrow n \]
In summary, a jump point is defined as meeting at least one of the following three conditions:

1. Node $x$ is the start or end point.
2. Node $x$ has at least one mandatory neighbor point.
3. When node $x$ moves in the diagonal direction, there are points satisfying conditions (1) and (2) in its horizontal and vertical directions.

**Path generation**

After jump point screening, the obtained jump points are added to the Openlist table of the $A^*$ algorithm for evaluation, and the evaluation function is shown in equation (4)

$$f(n) = h(n) + g(n)$$

where $f(n)$ is the evaluation function, $h(n)$ is the path cost from the current node to the target node, and $g(n)$ is the path cost from the start node to the current node. This algorithm uses the Euclidean distance between two points as the path of the cost.

Select the target node with the lowest cost value and add it to the Closelist table of the $A^*$ algorithm, then connect the nodes in the Closelist table and we can get the path.

The $A^*$ algorithm can only select eight adjacent directions for calculation in the next node selection, but the JPS algorithm can choose from jump points, and achieve movement across multiple node distances, which greatly improved the operating efficiency.

As shown in Figure 4, S is the start point and G is the target point. Firstly, record start point S as the jump point according to the jump point rule, search in the horizontal and vertical directions, and the search stops when encountering a map boundary or obstacle, after searching in horizontal and vertical directions, move a grid diagonally to continue the horizontal and vertical search. Repeat the above operation, when moving to point A find mandatory neighbor point B, and record the start point A and B as the jump point according to the jump point rule. Point B search down and find the goal point G, record the goal point G as a jump point, and stop the search. Finally, we can get the path through the relationship between the parent node and the child node.

**JPIACO algorithm**

**ACO algorithm**

ACO algorithm is to imitate the foraging behavior of ant colonies in nature. It obtains the optimal path by continuously updating the pheromone between colony and environment. The shorter path that the ant walks, the more concentration of pheromones left, the ant has a higher probability to choose the path, so there will be more and more pheromones on the path, thus forming an optimal path positive feedback strengthening mechanism.
In the ACO algorithm, roulette is generally used to select the next node, as shown in equation (5)

\[
P^k_{ij}(t) = \begin{cases} \frac{\tau_{ij}^k(t) \cdot \eta_{ij}^k(t)}{\sum_{s \in \text{allowed}_k} \tau_{is}^k \cdot \eta_{is}^k(t)} & s \in \text{allowed}_k \\ 0 & s \notin \text{allowed}_k \end{cases}
\]

(5)

where \( \tau_{ij}^k(t) \) is the pheromone concentration between node \( i \) and node \( j \), \( \eta_{ij} = 1/d_{ij} \) is the heuristic function and \( d_{ij} \) is the Euclidean distance; \( j \) is the next mobile node and \( s \) is an unprocessed node; \( \alpha \) and \( \beta \) are two constants, respectively, represent the importance of pheromone concentration and heuristic function; allowed\(_k\) is a collection of nodes that have not yet passed.

When the ACO algorithm completes a loop search, the pheromone on the path is updated globally, the calculation formula is shown in equations (6) and (7)

\[
\begin{align*}
\tau_{ij}(t + 1) &= (1 - \rho) \tau_{ij}(t) + \Delta \tau_{ij} \\
\Delta \tau_{ij} &= \sum_{k=1}^{m} \Delta \tau_{ij}^k \\
\Delta \tau_{ij}^k &= \begin{cases} 
Q/L_k & \text{find the path} \\
0 & \text{others}
\end{cases}
\end{align*}
\]

(6)

(7)

where \( \rho \) is the pheromone evaporation coefficient between 0 and 1, \( m \) is the number of ants, \( Q \) is a constant greater than 0, and \( L_k \) is the length of the path found by the ant \( k \).

**Improvement of ACO algorithm**

For the shortcomings of the ACO algorithm such as slow iteration speed and unsmooth path, modify the heuristic function of the ACO algorithm, introduce the reward and punishment factor from the Wolf pack colony algorithm, and limit the maximum and minimum ant pheromone concentration.

**Design of heuristic function.** The heuristic function of the ACO algorithm is reciprocal to the Euclidean distance from the current node to the target node, but in fact, the mobile robot will generate extra time when turning in motion, so the path will be selected with fewer turns when the path length is equal. Based on this, the cost of turns is added to the heuristic function to make the generate path smoother, heuristic function is shown in equations (8) and (9)

\[
\eta_{ij}(t) = \frac{1}{d_{ij} + c_{ij}}
\]

(8)

\[
c_{ij} = \omega \cdot \psi \cdot \theta
\]

(9)

where \( \omega \) (0 < \omega < 1) is the weight of turning cost, \( \psi \) is the factor converted to grid length, and \( \theta \) is the angle between the forward and backward motion directions of the current node.

**Improvement of initial pheromone.** The pheromone concentration of each node in the early stage of the ACO algorithm is the same, which leads to the problem of blind path search, poor convergence, and long search time. To address this problem, this algorithm first finds a series of jump points by the JPS algorithm, and these jump points produce a guiding effect in the early stage, which speeds up the previous iteration, the initial concentration value is shown in equation (10)

\[
Gau = \frac{Q}{L_{jps}} \cdot \frac{D_{sum}}{D_{jps}}
\]

(10)

where \( Gau \) is the initial pheromone concentration value at each jump point, \( L_{jps} \) is the path length found by the JPS algorithm, \( D_{sum} \) is the total number of points passed by the path, and \( D_{jps} \) is the number of jump points.

**Improvement in pheromone update.** For the shortcomings of the ACO algorithm in convergence speed slow, introduce reward and punishment factors in the Wolf pack algorithm.26 After each iteration, pheromone reward for the optimal path and penalty for the worst path. Furthermore, aim at the difference between pathfinding in early and later stages of iteration, conduct dynamic adjustment of pheromone concentration by increasing the reward and punishment factors in early iteration and decreasing the factors in the later iteration, because it’s necessary to quickly find a relative path in the early stage and increase the search range in the later stage. The formula is defined as follows

\[
\begin{align*}
\tau_{ij}(t + 1) &= (1 - \rho) \tau_{ij}(t) + \Delta \tau_{ij} + pau \\
\Delta \tau_{ij} &= \sum_{k=1}^{m} \Delta \tau_{ij}^k \\
\Delta \tau_{ij}^k &= \begin{cases} 
\frac{Q}{L_k + C_k} & \text{find the path} \\
0 & \text{others}
\end{cases} \\
pau &= \begin{cases} 
\left(1 - \frac{n}{N}\right) \cdot Q & \text{optimal path of this iteration} \\
\left(1 + \frac{n}{N}\right) \cdot Q & \text{worst path of this iteration} \\
0 & \text{others}
\end{cases}
\end{align*}
\]

(11)

where \( Pau \) is the reward and punishment factor, \( C_k \) is the total turn cost of ant \( k \), \( n \) and \( N \) are, respectively, the current number and the total number of iterations, other parameters are consistent with the ACO algorithm.

In the ACO algorithm, the pheromone concentration on one path is much higher than on other paths after multiple iterations, which may trap into the local optimum and lose
the global optimum solution. To avoid it, introduce the Max and Min system to limit it\textsuperscript{27}, the formula is shown in equation (12)

$$
\begin{align*}
\text{Taumax} \leq \text{Tau} \leq \text{Taumin} \\
\text{Taumin} / C_{20} \leq \text{Tau} \leq \text{Taumax} / C_{20}
\end{align*}
$$

(12)

where $\text{Tau}$ is the total pheromone concentration at any point on the map.

As shown in Algorithm 1, this is the pseudo-code of the JPIACO algorithm.

**Algorithm 1. JPIACO algorithm.**

Input: $x_{\text{start}}$, $x_{\text{goal}}$, $\alpha$, $\beta$, $k$, $m$, $Q$, $\rho$, $N_{\text{max}}$

Output: The best path

1. Build the map Model
2. Initialize the algorithm
3. Calculate the initial pheromone and distribute it on the map through Eq. (11)
4. for $i=1$ to $N_{\text{max}}$
5. for ant=1 to $k$
6. while node$_{\text{current}} \neq x_{\text{goal}}$
7. choose next node through Eq. (5)
8. end
9. end
10. Calculate pheromone concentration of the best path through Eq. (12)
11. Calculate pheromone concentration by Reward and punishment factor through Eq. (12)
12. Pheromone update through Eq. (11)
13. Pheromone concentration was limited through Eq. (14)
14. end
15. return the best path

As seen from Figures 6 to 8, the ACO algorithm finds the relatively optimal shortest path in the small environmental scale map, but with the increase of environmental scale, there are a lot of twists and turns in the path. However, the JPIACO algorithm can always find the relatively better shortest path. Moreover, the ACO algorithm finds the relative shortest path after 13 times in the environment of map 1, with the increase of the environment scale, the iterative process has not stabilized and the path is not optimal as we can see. But the JPIACO algorithm can find the relative optimal path in few times, it stabilized after 2 iterations in map 1, stabilized after 6 iterations in map 2, and stabilized after 22 iterations in map 3. It further confirms the effectiveness of the JPIACO algorithm.

**Simulation**

So as to verify the effectiveness of the proposed algorithm in path planning, compare it with the ACO algorithm through path length, iteration times, and number of turns. Experiments were run on an Intel i7-4720HQ cpu with 8G of RAM, and the programming language is MATLAB(R2018b). The ACO algorithm parameters are set according to the literature\textsuperscript{10,12,28} set according to the original parameters, and the parameters set in this article are shown in Table 1.

As shown in Figure 5, use map 1 (15*15), map 2 (30*30), and map 3 (50*50) in the literature\textsuperscript{7} as the environment for simulation experiments.

Figures 6(a), 7(a) and 8(a) are the optimal paths of the ACO algorithm under three maps: Figures 6(b), 7(b), and 8(b) are the optimal paths of JPIACO algorithm; Figures 6(c), 7(c), and 8(c) are the optimal path length iteration curve comparison of two algorithms.

**Table 1. The parameter setting.**

| Parameter | $N_{\text{max}}$ | $k$ | $\alpha$ | $\beta$ | $\rho$ | $Q$ |
|-----------|-----------------|-----|---------|--------|------|----|
| Value     | 50              | 30  | 1       | 18     | 0.3  | 100|

As seen from Figures 6 to 8, the ACO algorithm finds the relatively optimal shortest path in the small environmental scale map, but with the increase of environmental scale, there are a lot of twists and turns in the path. However, the JPIACO algorithm can always find the relatively better shortest path. Moreover, the ACO algorithm finds the relative shortest path after 13 times in the environment of map 1, with the increase of the environment scale, the iterative process has not stabilized and the path is not optimal as we can see. But the JPIACO algorithm can find the relative optimal path in few times, it stabilized after 2 iterations in map 1, stabilized after 6 iterations in map 2, and stabilized after 22 iterations in map 3. It further confirms the effectiveness of the JPIACO algorithm.

To verify the performance advantages of the JPIACO algorithm in path planning, compare it with the ACO algorithm and algorithm of literature\textsuperscript{10,12} to obtain the number of turns, minimum iteration, and optimal path length in three maps environments, as shown in Table 2.

As seen from Table 2, with the increase in the complexity of the environment, the ACO algorithm finds a larger and larger gap in the length of the path compared to the JPIACO algorithm, from a difference of 4 in map 2 to a difference of 48.14 in map 3, the number of turns and minimum iteration are the largest among the four algorithms in three maps expect in map 1. The algorithm
proposed in the literature\textsuperscript{10} can find the optimal solution in a few iterations, with the increase of environmental complexity, the optimal path length found in the environment of map 3 is 77.15, which is still insufficient compared with 71.64 found by the JPIACO algorithm, the number of turns is deficiency compare with this article, but for the minimum number of iterations, literature\textsuperscript{10} found the minimum iteration in map 3. The algorithm proposed in the literature\textsuperscript{12} can deal with deadlock better, but the path found in map 2 and map 3 is inferior to the JPIACO algorithm, the number of turns and minimum iteration among the four algorithms are inferior to this article. In terms of path accuracy and the number of turns, the algorithms proposed in this article find the best path and the least turns among the four algorithms, for the minimum number of iterations, this article is optimal between map 1 and map 2, and second only to the literature\textsuperscript{10} in map 3. What’s more, in terms of turning angle, the algorithm in this article is 45° turning
angle, compared with 90° turning angle and 135° turning angle existing in other algorithms, the cost of turning is smaller and more in line with the actual path planning application, which has a great practical value.

**Conclusion**

This article presents the JPIACO algorithm. Aiming at the problems of the ACO algorithm, such as slow early convergence speed and an excessive number of turning points, the JPS algorithm is added to obtain jump points by guiding a pathfinding function to distribute initial pheromone concentration values in the map. The turn cost factor, additional reward and punishment factors, and Max–Min ant system are added, and these improve the shortcomings of slow convergence speed and a lot of turns of the ACO algorithm. JPIACO algorithm is simulated under three kinds of maps and compared with the algorithm in literature, and the results show that the JPIACO algorithm has an improved effect, which can quickly find an effective path with high precision, few turning points, and small turning angles. Moreover, it is still effective in complex environments. The simulation results of different map sizes show that the JPIACO algorithm is generally adaptable to map scale and obstacle distribution structure.

Although JPIACO is a promising algorithm, the kinematic constraints must be considered if applied to robots. In addition, the performance of the algorithm in the dynamic environment should also be verified, and that’s what we’re going to do in the future.

**Declaration of conflicting interests**

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

**Funding**

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by National Key Research and Development Program of China (Grant No. SQ2020YFF0423771) and Zhejiang Science and Technology planning project (Grant No. 2020C01053).

**ORCID iD**

Tao Chen https://orcid.org/0000-0002-5172-7408

**References**

1. Zhao Y, Zheng Z and Liu Y. Survey on computational-intelligence-based UAV path planning. Knowledge-Based Syst 2018; 158: 54–64.
2. Hu B, Cao Z and Zhou M. An efficient RRT-based framework for planning short and smooth wheeled robot motion under kinodynamic constraints. IEEE Trans Indus Electr 2020; 68: 99.
3. Wang H. An improved timed elastic band (TEB) algorithm of autonomous ground vehicle (AGV) in complex environment. *Sensors* 2021; 21: 8312.

4. Zhu DQ and Yan MZ. Survey on the technology of mobile robot path planning. *Contr Decis* 2010; 25(7): 961–967.

5. Zhao X, Wang Z, Huang CK, et al. Mobile robot path planning based on an improved A* algorithm. *Robot* 2018; 40(6): 903–910.

6. Yuan CC, Wei Y, Shen J, et al. Research on path planning based on new fusion algorithm for an autonomous vehicle. *Int J Adv Robot Syst* 2020; 17(3): 172988142091123.

7. Kumar A, Kumar PB and Parhi DR. Intelligent navigation of humanoids in cluttered environments using regression analysis and genetic algorithm. *Arabian J Sci Eng* 2018; 43(12): 7655–7678.

8. Petrović M, Vuković N, Mitić M, et al. Integration of process planning and scheduling using chaotic particle swarm optimization algorithm. *Exp Syst Appl* 2016; 64: 569–588.

9. Wang P, Gao S, Li L, et al. Obstacle avoidance path planning design for autonomous driving vehicles based on an improved artificial potential field algorithm. *Energies* 2019; 12(12): 2342.

10. Zhang W, Ma Y, Zhao HD, et al. Obstacle avoidance path planning of intelligent mobile-based on improved firework-ant colony hybrid algorithm. *Contr Decis* 2019; 34(2): 335–343.

11. Dorigo M, Maniezzo V and Colomi A. Ant system: optimization by a colony of cooperating agents. *IEEE Trans Syst Man Cybern Pt B Cybern* 1996; 26(1): 29–41.

12. Sun GW, Su YX, Gu YC, et al. Path planning for unmanned surface vehicle based on improved ant colony algorithm. *Contr Decis* 2021; 36(4): 847–885.

13. Zhang S, Pu J and Si Y. An adaptive improved ant colony system based on population information entropy for path planning of mobile robot. *IEEE Access* 2021; 9: 99.

14. Zhang H, He L, Yuan L, et al. Mobile robot path planning using improved double-layer ant colony algorithm. *Contr Decis* 2022; 37(02): 303–313.

15. Khaled A and Farid K. Mobile robot path planning using an improved ant colony optimization. *Int J Adv Robot Syst* 2018; 15(3): 172988141877467.

16. Luo Q, Wang H, Zheng Y, et al. Research on path planning of mobile robot based on improved ant colony algorithm. *Neural Comput Appl* 2020; 32: 1555–1566.

17. Meng L, You X and Liu S. Multi-colony collaborative ant optimization algorithm based on cooperative game mechanism. *IEEE Access* 2020; 8: 99.

18. Tao Y, Gao H, Ren F, et al. A mobile service robot global path planning method based on ant colony optimization and fuzzy control. *Appl Sci* 2021; 11(8): 3605.

19. Ma Y, Mao Z, Wang T, et al. Obstacle avoidance path planning of unmanned submarine vehicle in ocean current environment based on improved firework-ant colony algorithm. *Comput Electr Eng* 2020; 87: 106773.

20. Dai X, Long S, Zhang Z, et al. Mobile robot path planning based on ant colony algorithm with A* heuristic method. *Front Neurorobot* 2019; 13: 15.

21. Zhu S, Zhu W, Zhang X, et al. Path planning of lunar robot based on dynamic adaptive ant colony algorithm and obstacle avoidance. *Int J Adv Robot Syst* 2020; 17(3): 4149–4171.

22. Yu M. A solution of TSP based on the ant colony algorithm improved by particle swarm optimization. *Discrete Contin Dynam Syst S* 2018; 12(4): 979–987.

23. Lei W, Luo C, Li M, et al. Trajectory planning of an autonomous mobile robot by evolving ant colony system. *Int J Robot Auto* 2017; 32(4): 1500–1515.

24. Saenphon T, Phimoltares S and Lursinsap C. Combining new fast opposite gradient search with ant colony optimization for solving traveling salesman problem. *Eng Appl Artif Intell* 2014; 35: 324–334.

25. Li SD, You XM and Liu S. Dynamic hierarchical dual-morphic ant colony algorithm based on artificial bee colony algorithm. *Comput Eng Appl* 2020; 56(12): 37–46.

26. Wu HS, Zhang FM and Wu L. A new swarm intelligence algorithm—wolf pack algorithm. *Syst Eng Electr Technol* 2013; 35(11): 2430–2438.

27. Stützle T and Hoos HH. MAX–MIN ant system. *Fut Gen Comput Syst* 2000; 16: 889–914.

28. Shi EX, Chen MM, Li J, et al. Research on method of global path planning for mobile robot based on the ant-colony algorithm. *Trans Chinese Soc Agricul Mach* 2014; 45(6): 53–57.