Electronics and Electrical Engineering

AIMS Electronics and Electrical Engineering, 9(4): 423–447.
DOI: 10.3934/electreng.2025020
Received: 16 May 2025
Revised: 24 July 2025
Accepted: 30 July 2025
Published: 06 August 2025

https://www.aimspress.com/journal/ElectrEng

Research article

NI-MOD-P-RRT*: A threshold-free sampling-based path planning algorithm for unmanned surface vehicles in dynamic environments

Yunlai Fu[1,2], Wei Wang[1,2,*], Yongqi Zhang[1,2], Yan Liu[1,2] and Haoran Li[1,2]

[1] College of Information Engineering, Dalian Ocean University, Dalian 116023, China
[2] Dalian Key Laboratory of Smart Fisheries, Dalian Ocean University, Dalian 116023, China

*** Correspondence:** Email: ww_wangwei@dlou.edu.cn.

Abstract: In this paper, we propose a threshold-free multi-objective dynamic rapidly-exploring random tree algorithm (NI-MOD-P-RRT*) for unmanned surface vehicles in unknown dynamic environments. The algorithm consists of an initial path generation process and a path replanning process. Its major features include: Addressing the issues of slow search speed and excessive inflection points in initial paths of the P-RRT* algorithm by integrating target-biased sampling and constraining the expansion angle of the random tree during sampling, optimizing connection constraints to reduce search time; after obtaining the initial path, employing improved triangle inequality and B-spline interpolation for path pruning and smoothing to generate an optimized path, with node information stored as prior knowledge; building upon this, designing a topology-sorting-based method to select optimal substitute nodes for path replanning when the current path becomes infeasible. Simulation experiments in various environments demonstrated that the proposed algorithm decreases path length, search time, and iteration counts, while quickly identifying substitute nodes to circumvent newly detected obstacles when needed.

Keywords: rapidly-exploring random tree (RRT); dynamic environments; unmanned surface vehicles; triangle inequality; topology-sorting

1. Introduction

Unmanned surface vehicles (USV) are widely used in various aspects of aquaculture, including intelligent feeding, drug delivery, water quality monitoring, and surface waste cleaning [1]. Path

planning technology remains a key technique in USV navigation, referring to the process of generating an optimal or suboptimal collision-free trajectory from start point to goal within a mission area based on movement requirements and environmental information while adhering to specified metrics (shortest distance, minimal time, lowest energy consumption, etc.) [2].

Currently, the more common path planning methods include geometry-based A* algorithm [3], D* algorithm [4], bio-inspired algorithms such as genetic algorithm [5], ant colony algorithm [6], particle swarm optimization [7], and sampling-based methods like probabilistic roadmap (PRM) [8] and rapidly-exploring random tree (RRT) [9]. Geometry-based search algorithms require building an environmental model and using heuristic functions for search, typically finding optimal or suboptimal paths but with high computational complexity, making them unsuitable for environments with numerous obstacles. Bio-inspired algorithms employ optimization techniques from artificial intelligence, simulating natural biological or physical phenomena to find global optimal or suboptimal solutions in environmental maps, usually demonstrating good robustness and adaptability, but with slower convergence speeds and requiring numerous parameter adjustments [10]. Sampling-based algorithms generate random sampling points in environmental maps, connecting them into a road network graph for subsequent searching, generally unaffected by map dimensionality or environmental complexity, but cannot guarantee optimal or suboptimal solutions and may carry collision risks. Compared to other algorithms, the RRT algorithm is widely used due to its few parameters, strong search capability, and applicability to various complex environments. For example, Ganesan, S proposed a Hybrid-RRT* algorithm for the autonomous navigation of robots [11]. Yang, Q, aiming at the continuous motion planning problem of redundant tiling manipulators in complex environments, designed an offline redundancy optimization algorithm based on the improved RRT*. This algorithm maximizes the utilization of kinematic redundancy to obtain smooth joint trajectories without collisions and singularities [12]. Fan, J applied the RRT* algorithm to the field of unmanned aerial vehicles (UAV) to solve the problem of UAV path planning in complex environments [13]. Liu, Y improved the RRT* algorithm through methods such as map preprocessing, nearest node selection optimization, connection constraint optimization, and triangle inequality path optimization, making it applicable to maps in narrow exit environments [14]. Gu, Q proposed a new method of PI-DP-RRT aiming at problems such as slow convergence, excessive turning points, and uneven path generation of RT-related algorithms in ship path planning. This method combines the Prior Automatic Identification System (AIS) information and Douglas-Peucker (DP) compression for ship path planning [15]. Zhou, Y applied the RRT* algorithm to the lifting path planning of mobile cranes to improve the efficiency and safety of crane lifting practices [16]. However, the RRT algorithm has some drawbacks, such as random sampling leading to numerous invalid points, non-smooth paths, and non-optimal paths. To address these shortcomings, many domestic and foreign scholars have improved the algorithm for path planning in static environments. In 2011, Karaman et al. [17] proposed the asymptotically optimal RRT* algorithm, further optimizing paths by improving parent node selection and rewiring. Building on this, Gammell et al. [18] introduced Informed-RRT*; Islam et al. [19] proposed the RRT*-smart algorithm; Jordan et al. [20] developed the Quick-RRT* algorithm. Although these improvements used less planning time and path length in static environments, none altered the blind sampling approach of the RRT* algorithm. Therefore, Ahmed et al. [21] combined the artificial potential field method with the RRT* algorithm to propose the P-RRT* algorithm, which directs sampling points toward the goal, but P-RRT* suffers from slow search speeds and poor initial solution quality. Moreover, the challenge in

path planning lies in dynamic environment path planning. While there are many mature methods for static environments, extending these to dynamic environments is a key focus of path planning research.

To address the issues of excessive redundant points in the search process, low initial path quality, and inability to adapt to dynamic environments, a new threshold-free multi-objective RRT* path planning algorithm (NI-MOD-P-RRT*) is proposed. This algorithm divides the dynamic path planning problem into two parts: Initial path generation and path replanning during navigation. The solution handles each step through two independent processes. First, an improved RRT* algorithm is introduced to generate an optimized path through pruning and smoothing based on the initial path. Then, a path replanning scheme is proposed to avoid collisions with unknown obstacles. Overall, the major contributions of this paper can be summarized as follows.

- In the preprocessing stage, the map undergoes envelope processing, and a strategy for expanding from the goal to the start point is designed for P-RRT*, thereby storing the cost from any tree node to the target location as prior information.
- During the sampling phase, the integration of goal-biased sampling, restricted sampling angles, and optimized distance threshold connection constraints reduces the generation of unnecessary nodes and improves search speed.
- After initial path generation, the improved triangular inequality and B-spline are used to optimize the initial path, enhancing its quality.
- During navigation, a reconnection scheme is designed to optimize the tree structure when obstacles are unknown.
- A multi-objective path replanning method based on a greedy algorithm is proposed, which selects the best alternative node when the path is blocked by obstacles during navigation.

The remainder of this paper is organized as follows. In Section 2, we introduce the algorithms related to this work and the problem of dynamic path planning. The proposed NI-MOD-RRT* algorithm is described in Section 3. In Section 4, we present simulation comparisons with other algorithms in static environments and dynamic replanning under various conditions. The conclusions are provided in Section 5.

2. Problem description and related fundamental algorithms

2.1. Path planning problem description

Let $X \subseteq R^d$ be the bounded task space of the path planning problem, where d represents the dimension of the task space, $\mathrm{d} \subseteq N$ and $d \geq 2$. The task space and the obstacle regions within it are denoted by X and X_{obs}, respectively, and $X_{free} = X / X_{obs}$ represents the obstacle-free search area in the space. $X_{start} \subseteq X_{free}$ is the starting point of the USV, $X_{goal} \subseteq X_{free}$ is the target point, and the continuous function $\mathrm{s}:[0,1] \to R^d$ represents the trajectory of the USV. If for all $\tau \in [0,1]$, there is $s(\tau) \in X_{free}$, $s(0) = X_{obs}$, and $s(1) = X_{goal}$, then the path planning problem can be defined by $(X_{free}, X_{start}, X_{goal})$, expressed as calculating a continuous collision-free path from the starting point X_{start} to the target point X_{goal} in X_{free} through a path planning algorithm as the solution to the problem [22].

For the optimal path planning problem of the USV, it is typically defined as searching for the shortest path among feasible paths, where $c(s^*) = \min c(s)$: s is a feasible path and $c(s)$ is the length of the path.

In this section, the definition of the dynamic path planning problem is given. Dynamic path planning refers to a scenario where the positions of some obstacles in the initial map are known, while others are unknown. The USV detects unknown obstacles within a certain detection range using its onboard sensors (e.g., LiDAR), and its onboard positioning device can accurately determine its own location. The working environment is shown in Figure 1.

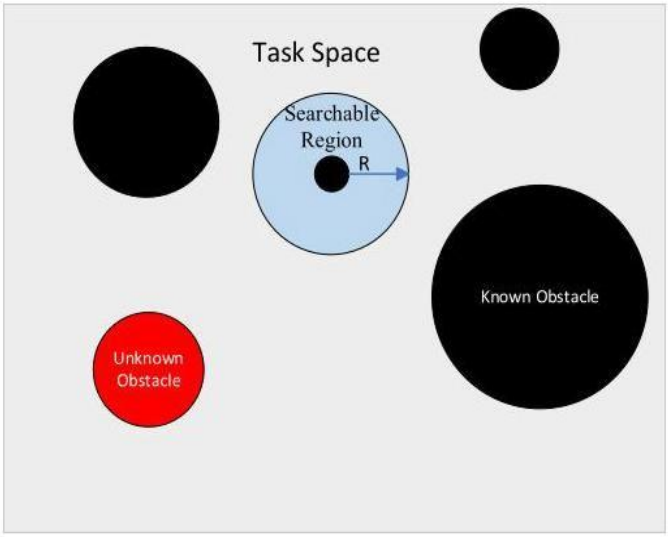


Figure 1. Working environment of the unmanned surface vehicle.

2.2. RRT algorithm*

The RRT* algorithm is an improvement over the RRT algorithm, with both sharing similar fundamental principles. The key enhancements in RRT* lie in the selection of parent nodes and tree rewiring. Before introducing RRT*, let us briefly describe the RRT algorithm: RRT explores the space via a tree $G = (V, E)$ starting from X_{start}, where V represents a set of nodes and E denotes the collection of connection relationships between nodes. Tree expansion consists of multiple iterations. In each iteration, a random node X_{rnd} is generated from X_{free}, the nearest node $X_{nearest}$ to X_{rnd} is identified in the tree, and a new node X_{new} is obtained by extending a fixed step size from $X_{nearest}$ toward X_{rnd}. If the connection between X_{new} and $X_{nearest}$ is collision-free with obstacles, X_{new} is added to the tree; otherwise, the iteration repeats. These steps are repeated until a feasible path from start to goal is found. Next, we present the pseudocode for RRT* as shown in Algorithm 1.

The primary difference between RRT* and RRT lies in the optimization modules *ChooseParent* and *Rewire*. In Algorithm 2, *ChooseParent* aims to replace the candidate parent node $X_{nearest}$ of X_{rnd} by searching X_{nears} (a set of nodes within a circle centered at X_{rnd} with radius R_{near}) for node X_{parent}, thereby reducing the path cost from X_{start} to X_{rnd}.

Algorithm 1 RRT*

Input: $X_{start}, X_{goal}, X_{obs}, Maxiter, R_{near}$

Output: $G=(V,E)$

1: $V \leftarrow \{X_{start}\}; E \leftarrow \varnothing;$

2: **for** $i=1$ **to** $Maxiter$ **do**

3: $X_{rnd} \leftarrow Sample(iter);$

4: $X_{nearest} \leftarrow Nearst(V, X_{rnd})$;

5: $X_{new} = Steer(X_{nearest}, X_{rnd});$

6: **if** $noCollision(X_{nearest}, X_{new}, X_{obs})$ **then**

7: $X_{nears} \leftarrow Near(V, X_{new}, R_{near});$

8: $X_{parent} \leftarrow ChooseParent(X_{nears}, X_{new}, X_{nearest});$

9: $V \leftarrow V \cup \{X_{new}\};$

10: $E = E \cup \{(X_{prnd}, X_{new})\};$

11: $G \leftarrow \mathrm{Re}wire(G, X_{new}, X_{nears});$

12: **if** $IsNearGoal(X_{new}, X_{goal})$ **then**

13: **if** $noCollision(X_{new}, X_{goal})$ **then**

14: **return** $G \leftarrow (V,E);$

15: **end if**

16: **end if**

17: **end if**

18: **end for**

19: **Return** $G=(V,E)$;

Algorithm 2 *ChooseParent*

Input: $X_{nears}, X_{new}, X_{nearest}$

Output: X_{parent}

1: $X_{parent} \leftarrow X_{nearest};$

2: $C_{\min} \leftarrow \mathrm{Cos}t(X_{nearest}) + Dist(X_{nearest}, X_{new});$

3: **for** each $X_{near} \in X_{nears}$ **do**

4: $C \leftarrow \mathrm{Cos}t(X_{near}) + Dist(X_{new}, X_{near});$

5: if $C < C_{\min}$ then

6: **if** $noCollision(X_{near}, X_{new})$ **then**

7: $X_{parent} \leftarrow X_{near};$

8: $C_{\min} \leftarrow C;$

9: **end if**

10: **end if**

11: **end for**

12: **Return** X_{parent}

Similarly, the purpose of *Rewire* in Algorithm 3 is to use the new node X_{new} as a parent, compute the path cost to other nodes within the circular region, and reconnect those nodes to X_{new} if the new path cost is lower than their original path cost.

Algorithm 3 *Rewire*

Input: G, X_{new}, X_{nears}

Output: $G=(V,E)$

1:**for** each $X_{near} \in X_{nears}$ **do**

2: **if** $\mathrm{Cost}(X_{near}) > \mathrm{Cost}(X_{new}) + Dist(X_{near}, X_{new})$ **then**

3: **if** $noCollision(X_{near}, X_{new})$ **then**

4: $E \leftarrow \left(E \setminus \left\{\left(Parent(X_{near}), X_{near}\right)\right\}\right) \cup \left\{\left(X_{near}, X_{new}\right)\right\}$;

5: **end if**

6: **end if**

7:**end for**

8:**Return** $G=(V,E)$;

To clarify the algorithmic steps, explanations of the pseudocode are provided below.

Sample: Returns a randomly sampled point in X_{free} space.

Nearst: Returns the node in the random search tree closest to the random point, using the Euclidean distance function.

Steer: Returns a new node generated by extending a fixed step size from the nearest point $X_{nearest}$ toward the random point X_{rnd}.

noCollision: Checks whether obstacles exist between two points.

Near: Returns the set of all nodes within a circle centered at X_{new} with radius R_{near}.

C: Represents the cost of the path from the start node to the current node.

Dist: Computes the Euclidean distance between two points.

IsNearGoal: Determines whether the distance between the new node and the goal is smaller than the expansion step size.

*2.3. P-RRT**

The P-RRT* algorithm is an improved version based on the RRT* algorithm. By introducing the artificial potential field (APF) method, it optimizes the generation direction of new nodes and enhances search efficiency. P-RRT* employs the random gradient descent (RGD) method, utilizing the attractive force of the potential field to guide random sampling points toward the goal and effectively reducing iteration counts. Algorithm 4 details the P-RRT* pseudocode.

Algorithm 5 elaborates on the RGD process. Under the attraction of the goal point X_{goal}, the random sample X_{rnd} is adjusted to obtain the refined sample X_{prnd}. The *NearestObstacle* procedure calculates the shortest distance from X_{prnd} to the obstacle space X_{obs}. Three parameters k, λ, and d_{obs}^{*} are used in RGD. Here, k represents the iteration count, λ denotes expansion toward the goal direction, and d_{obs}^{*} is a minimal distance from X_{obs}. These parameters require manual tuning, though parameter optimization is omitted here. This paper sets $k = 40$, $\lambda = 0.05$, and $d_{obs}^{*} = 0.05$ (the parameters are from Reference [23]).

Algorithm 4 P-RRT*

Input: $X_{start}, X_{goal}, X_{obs}, Maxiter, R_{near}$

Output: $G=(V,E)$

$V \leftarrow \{X_{start}\}; E \leftarrow \varnothing;$

for $i=1$ **to** $Maxiter$ **do**

  $X_{rnd} \leftarrow Sample(iter)$;

  $X_{prnd} \leftarrow RGD(X_{rnd}, d^*_{obs}, \lambda)$;

  $X_{nearest} \leftarrow Nearst(V, X_{prnd})$;

  $X_{new} = Steer(X_{nearest}, X_{prnd})$;

  if $noCollision(X_{nearest}, X_{new}, X_{obs})$ **then**

    $X_{nears} \leftarrow Near(V, X_{new}, R_{near})$;

    $X_{parent} \leftarrow ChooseParent(X_{nears}, X_{new}, X_{nearest})$;

    $V \leftarrow V \cup \{X_{new}\}$;

    $E = E \cup \{(X_{parent}, X_{new})\}$;

    $G \leftarrow \mathrm{Re}wire(G, X_{new}, X_{nears})$;

    if $IsNearGoal(X_{new}, X_{goal})$ **then**

      if $noCollision(X_{new}, X_{goal})$ **then**

        return $G=(V,E)$;

      end if

    end if

  end if

end for

Return $G=(V,E)$;

Algorithm 5 RGD

Input: $X_{rnd}, d^*_{obs}, \lambda$

Output: X_{prnd}

$X_{prnd} \leftarrow X_{rnd}$

for $n=1$ to k **do**

  $\vec{F}_{att} \leftarrow (X_{goal} - X_{prnd})$;

  $d_{\min} \leftarrow NearestObstacle(X_{obs}, X_{prnd})$;

  if $d_{\min} < d^*_{obs}$ **then**

    Return X_{prnd}

  else

    $X_{prnd} \leftarrow X_{prnd} + \lambda \dfrac{\vec{F}_{att}}{|F_{att}|}$

  end if

end for

Return X_{prnd}

3. NI-MOD-P-RRT* algorithm

To address the aforementioned issues, we propose a novel NI-MOD-P-RRT algorithm. Its overall workflow is illustrated in Figure 2, where blue dashed boxes indicate improved components (section 3.3 involves constraint reduction and is unmarked). First, a map preprocessing method is introduced, narrowing the exploration space of the random tree via goal-biased sampling, limited sampling angles, and optimized connection constraints, thereby improving path search efficiency. Second, path length is optimized through enhanced triangle inequality pruning and B-spline smoothing. Third, a path replanning procedure for navigation is proposed.

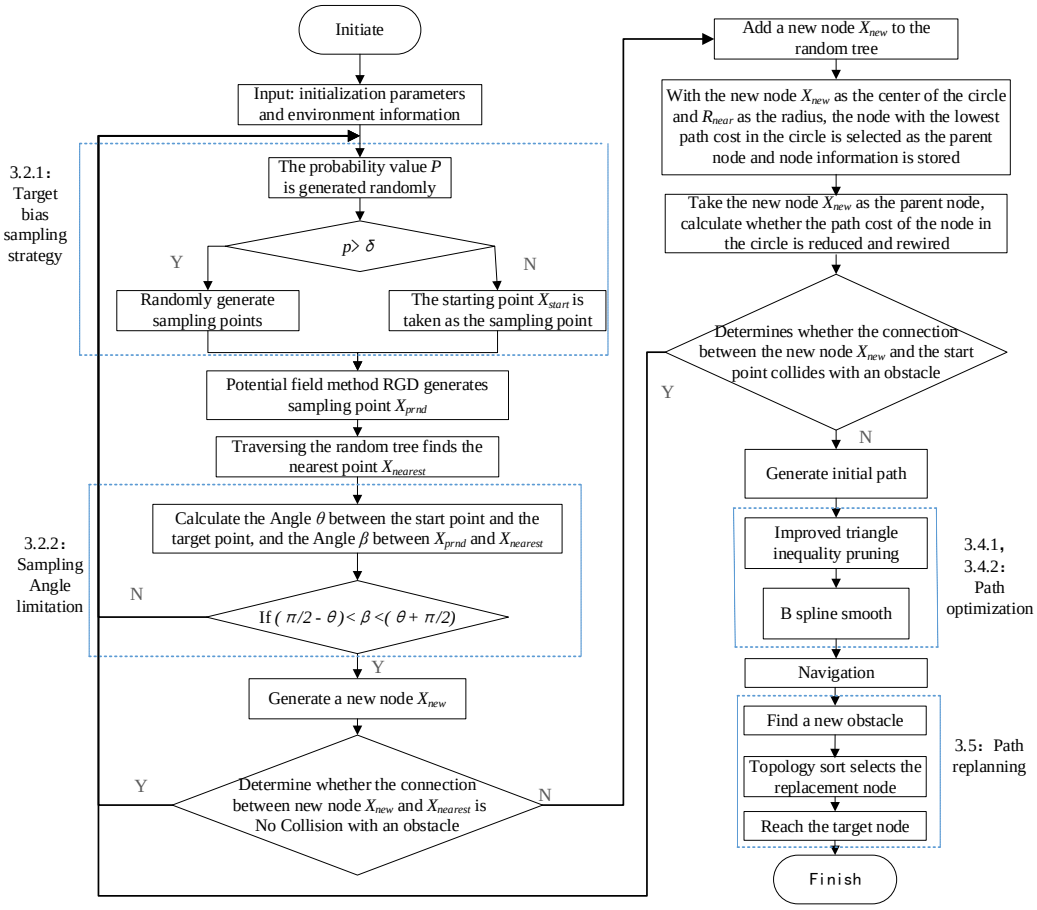


Figure 2. Overall algorithm flowchart.

3.1. Obstacle preprocessing

Conventional P-RRT* treats USV as point masses for path planning. Without reserving clearance between the point mass and obstacles, generated paths may cause collisions in practice. Thus, this issue necessitates resolution. As shown in Figure 3, obstacle enveloping is performed considering the USV geometry.

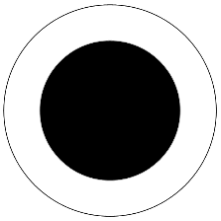


Figure 3. Obstacle enveloping.

3.2. Sampling strategies

3.2.1. Goal-biased sampling

Although the P-RRT* algorithm introduces the potential field method in the sampling phase to bias sampling points toward the starting point, the time to search for the optimal path remains lengthy. To decrease search time, a goal-biased sampling strategy is incorporated in each random sampling iteration, where a random probability determines whether the sampled point X_{rnd} in the task space is a random point or the starting point. Goal-biased sampling involves setting a reference value $\delta(0<\delta<1)$ during sampling, then generating a random probability $p(0<p<1)$ before each round of random sampling. If $0<p<\delta$, the starting point X_{start} is selected as the sampling point; otherwise, random sampling is performed in the space, as shown in Equation (1), δ = 2% (the parameters are from Reference [24]).

$$\begin{cases} X_{rnd} = random(X_{\text{free}}) & \delta < p \\ X_{rnd} = X_{\text{start}} & 0 < p < \delta \end{cases} \tag{1}$$

3.2.2. Sampling angle restriction

Due to the randomness of the P-RRT* algorithm's sampling, some invalid sampling points may arise, leading to resource wastage. To address this issue, we restrict the sampling angle, reducing the generation of invalid sampling points and accelerating the search process. First, the expansion angle β between the random sampling point X_{prnd} and the nearest node $X_{nearest}$ is calculated. Then, the angle θ between the starting point X_{start} and the goal point X_{goal} is computed, representing the ideal direction of the global optimal path. If β falls within the $\left[\frac{\pi}{2}-\theta,\frac{\pi}{2}+\theta\right]$ range, a new node is generated; otherwise, resampling occurs, as illustrated in Figure 4. This ensures that the expansion direction of the new node always revolves around the direction of the target point, avoiding completely reverse or irrelevant random sampling, thereby reducing ineffective exploration.

3.3. Optimization of connection constraints

As shown in Algorithm 4, the P-RRT* algorithm terminates branch iteration under two constraints: first, the Euclidean distance between the new node X_{new} and the goal point X_{goal} is less than the expansion distance; second, the line connecting the new node X_{new} and the goal point

X_{goal} collides with obstacles. Due to the first constraint, the new node X_{new} attempts to connect to the goal point only when it is within the expansion distance. However, when the new node X_{new} is far from the goal point X_{goal} and no obstacles exist between them, the line segment formed by the two points can serve as a feasible path for unmanned vessels. In such cases, iteration can terminate immediately to generate an initial path without waiting for the new node to approach the goal point. Removing this constraint often enables the random tree to terminate iteration earlier, thereby improving the algorithm's initial path-solving speed, as depicted in Figure 5.

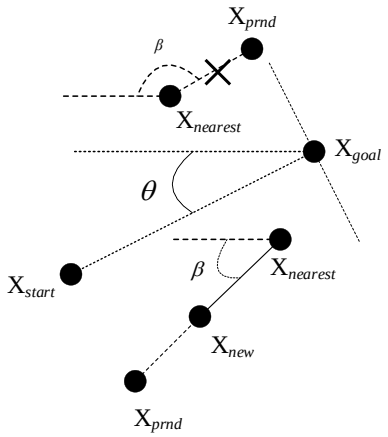


Figure 4. Expansion angle restriction.

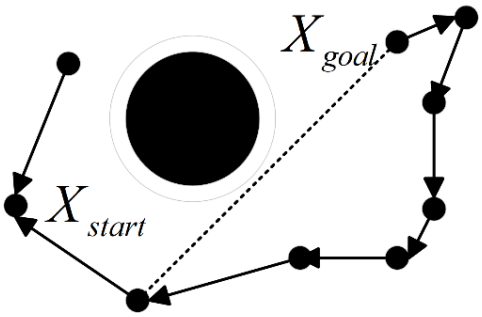


Figure 5. Planning diagrams with and without constraints.

3.4. Path optimization

3.4.1. Improved triangle inequality pruning

Due to the random sampling of P-RRT*, the generated path tends to be tortuous with redundant points. Triangle inequality pruning can not only eliminate unnecessary redundant points but also cut the path cost. However, traditional triangle inequality pruning merely removes redundant nodes, resulting in longer paths as they stay farther from obstacles. We improve the traditional triangle inequality by deleting redundant nodes while generating new nodes near obstacles to further shorten the path. The enhanced triangle inequality pruning process is as follows: First, check whether the line connecting the start point X_{start} and the target point X_{goal} collides with obstacles. If no collision occurs, remove all intermediate nodes and generate a direct path. If a collision is detected, skip the next node X_1 from the start point X_{start} and connect it directly to node X_2. If the connection

passes the collision detection, remove node X_1. Then, attempt to connect X_{start} with X_3. If the collision check fails, use the bisection method near obstacles on the lines of (X_{start}, X_2) and (X_2, X_3) to create a new node X_{creno}, and connect X_{start} to the new node X_{creno}. Next, simplify the path by treating X_{creno} as the new start point until the start point connects to the target point. The pseudocode for creating new nodes is shown in Algorithm 6, where X_i is the node before the collision when connecting to the start point.

Algorithm 6 *CreatNode*

Input: $G, X_{\text{cur}}, X_i, D_{dichotomy}$

Output: X_{creno}

$X_{all} \leftarrow X_i$;
if $X_i \neq X_{goal}$ **then**
  $X_{for} \leftarrow X_{(i+1)}$
  while $Dist(X_{all}, X_{for}) > D_{dichotomy}$ **do**
    $X_{mid} \leftarrow (X_{all} + X_{for})/2$;
    if $noCollision(X_{cur}, X_{mid})$ **then**
      $X_{all} \leftarrow X_{mid}$;
    else
      $X_{for} \leftarrow X_{mid}$
    end if
  end
  $X_{for} \leftarrow X_{cur}$
  while $Dist(X_{all}, X_{for}) > D_{dichotomy}$ **do**
    $X_{mid} \leftarrow (X_{all} + X_{for})/2$;
    if $noCollision(X_{mid}, X_{(i+1)})$ **then**
      $X_{all} \leftarrow X_{mid}$;
    else
      $X_{for} \leftarrow X_{mid}$
    end if
  end
end
if $X_{all} \neq X_i$ **then**
  $X_{credo} \leftarrow X_{all}$
else
  $X_{credo} \leftarrow \varnothing$;
end
Return X_{credo}

3.4.2. B-spline smoothing

Although triangle inequality pruning shortens the path length and reduces unnecessary nodes, the resulting path contains certain inflection points that do not conform to the motion patterns of

unmanned vessels. Therefore, further smoothing is required. Common path smoothing methods include gradient descent, Bézier curves [25], cubic splines [26], and B-spline curves [27]. Since B-spline curves exhibit continuous curvature, including at the junctions between curve segments, they are widely used in path smoothing. Thus, we adopt the B-spline method for smoothing. Given control points $P_0, P_1, \cdots, P_n$, which shape the curve, the expression for a k-degree B-spline is:

$$P(t) = \sum_{i=0}^{n} B_{i,k}(t) P_i \tag{2}$$

In Equation (2), $B_{i,k}(t)$ is the i-th k-order B-spline basis function, and the parameter t is a non-decreasing sequence of knot vectors. The basis functions are derived from the de Boor-Cox recurrence relation:

$$B_{i,k}(t) = \begin{cases} 1 & t_i < t < t_{i+1} \\ 0 & \text{Other} \end{cases} \qquad k=1 \tag{3}$$

$$B_{i,k}(t) = \frac{t - t_i}{t_{i+k-1} - t_i} B_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} B_{i+1,k-1}(t) \tag{4}$$

By sequentially connecting each segment of $B_{i,k}(t)$ with k-degree B-spline curves, the complete B-spline curve is formed.

The optimization results are shown in Figure 6: the blue dashed line represents the pre-optimization path, while the red curve indicates the optimized path.

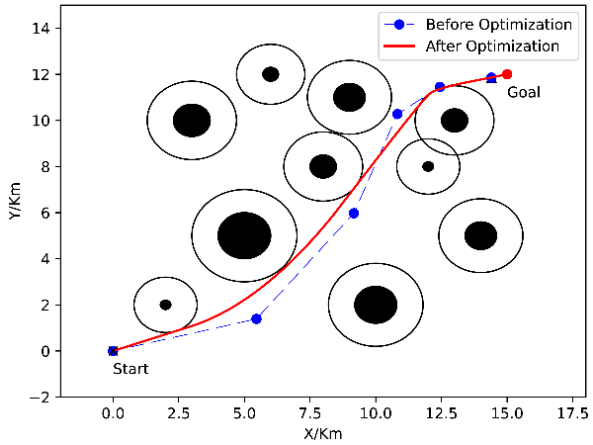


Figure 6. Comparison of paths before and after optimization.

3.5. Path replanning

During navigation, the unmanned vessel dynamically updates its map based on detected environmental information. Furthermore, the state tree optimizes its structure to ensure its nodes and edges avoid collisions with newly detected obstacles. The reconnection process is illustrated in Figure 7, where red denotes new obstacles and blue arrows indicate new connections. Moreover, collided nodes and vertices are deleted. Deletion may create isolated nodes, which will select new parent nodes according to Algorithm 2. For isolated node X_{rop}, a new parent node is reselected based on Algorithm 2.

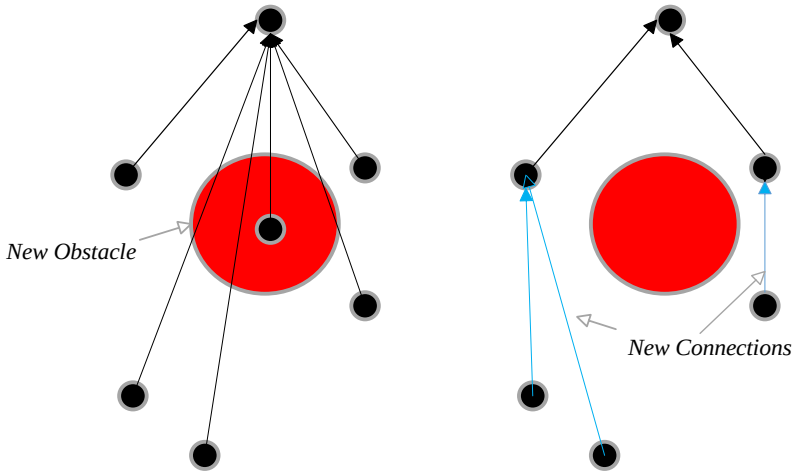


Figure 7. Node reconnection after detecting new obstacles.

When the unmanned vessel identifies new obstacles blocking its planned path, it searches the state tree for alternative nodes to circumvent them. As shown in Figure 8, assuming the vessel's current position is X_{curr}, the candidate nodes (marked in blue) represent the searchable range, excluding obstacle directions. Two metrics, path cost C_{candia} and angle A_{candia}, are used to evaluate candidate node X_{candia}. Here, A_{candia} is the steering angle between X_{curr} and X_{candia} (denoted by ρ in Figure 8), while C_{candia} is the total path length from X_{curr} to X_{goal} via X_{candia}, calculated as:

$$C_{candia} = \mathrm{Cos}t(X_{candia}) + Dist(X_{curr}, X_{candia}) \tag{5}$$

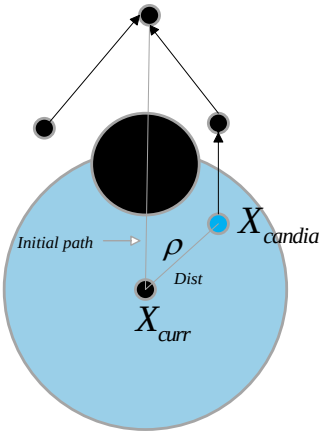


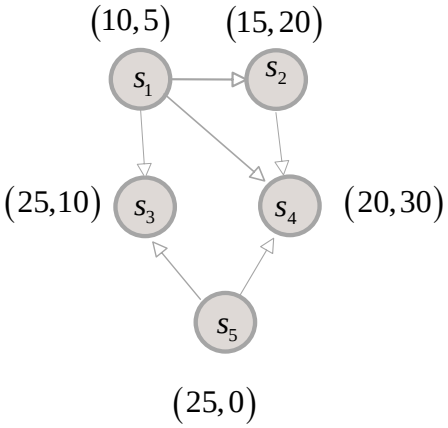
Figure 8. Candidate nodes in path replanning.

Dozens to hundreds of candidate nodes may exist within the selection range. The greedy algorithm constructs near-optimal connections by iteratively comparing and selecting nodes with minimal (optimal) path length and angle, following this principle [28]:

For vectors $s = [s_1, s_2, \cdots, s_n]$ and $y = [y_1, y_2, \cdots, y_n]$, if for all i, $s_i \le y_i$ and for at least one j, $s_i < y_j$, where $1 \le i, j < n$, then vector s dominates vector y. Therefore, a directed edge $s \to y$ is established (indicating that s "precedes" y in some sense). For example, $s = (10,5)$, $y = (15,10)$, and $z = (9,6)$ are three target state vectors. Both s and z dominate y, but no precedence relationship exists between them.

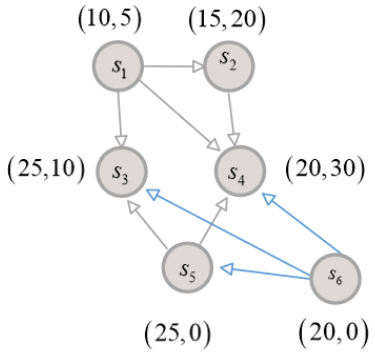
In this path replanning approach, the metrics for target vectors is path cost C_{candia} and angle A_{candia}. Using a greedy algorithm, candidate nodes can be prioritized to construct a directed acyclic state expansion graph for topological sorting. For every two nodes, if one dominates the other, an arrow is drawn; nodes without mutual dominance remain unconnected, as shown in Figure 9. Thus, the in-degree of node s, s represents the number of nodes dominating it, while the out-degree of node s corresponds to the number of nodes it dominates. Nodes with zero in-degree are locally optimal. Typically, multiple locally optimal nodes exist in the greedy algorithm, and the one with the highest out-degree is selected as the substitute. If ties persist, the node with the smallest angle A_{candia} is chosen, considering the motion patterns of unmanned vessels.

In Figure 9, the in-degree and out-degree of all nodes are listed. s_1 is the optimal state with an in-degree of 0 and out-degree of 3.



Node	I-D	O-D
s_1	0	3
s_2	1	1
s_3	2	0
s_4	3	0
s_5	0	2

Figure 9. Illustration of the topological sorting graph.



Node	I-D	O-D
s_1	0	3
s_2	1	1
s_3	3	0
s_4	4	0
s_5	1	2
s_6	0	3

Figure 10. Topological sorting graph after adding s_6.

Figure 10 illustrates the graph after adding a new node s_6. s_6 dominates s_3, s_4, and s_5, and no node dominates s_6, giving it an in-degree of 0 and out-degree of 3. Now, s_1 and s_6 share identical in/out-degrees, but s_6 has a smaller angle than s_1. Thus, s_6 is selected as the best substitute node.

Algorithm 7 Path Replanning

While $Not\,\mathrm{Re}achGoal$ **do**
  if $FindNewObstacle$ **then**
    $X_{curr} \leftarrow CurrentNode$;
    $X_{candias} \leftarrow \varnothing$;
    $X_{nears} \leftarrow Near(V, X_{curr}, R)$;
    for $X_{near} \in X_{nears}$ **do**
      if $noCollision(X_{curr}, X_{near}, X_{obs})$ **then**
        $X_{candias} \leftarrow X_{candias} \cup X_{near}$;
      end if
    end for
    $X_{next} \leftarrow G\text{reedy Algorithm}(X_{candias})$;
  end if
  $X_{curr} \leftarrow X_{next}$;
end while

3.6. Process description of the NI-MOD-P-RRT algorithm*

Based on the above analysis, the path planning process of the NI-MOD-P-RRT* algorithm is obtained. First, environmental information is acquired to generate a random probability p. It is determined whether p is greater than the probability threshold δ. If so, random sampling is performed; otherwise, the starting point is used as the sampling point. The potential field method RGD is employed to generate a sampling point X_{prnd}. It is then checked whether the expansion angle β between the sampling point X_{prnd} and the nearest node $X_{nearest}$ falls within the sampling angle limit range of the angle θ between the target point X_{goal} and the starting point X_{start}. If not, the iteration is repeated; otherwise, a new node X_{new} is generated. Collision detection is performed between node X_{new} and node $X_{nearest}$. If a collision occurs, the iteration is repeated; otherwise, the new node X_{new} is added to the random tree for reselecting the parent node and rewiring, and the distance to the target point is calculated. It is determined whether node X_{new} collides with the starting point X_{start}. If a collision occurs, the iteration is repeated; otherwise, the iteration ends, and a feasible path is generated. Subsequently, the improved triangle inequality and B-spline are used for pruning and smoothing to optimize the path for controlling the unmanned vessel's movement. If the route is blocked by unknown obstacles, the greedy algorithm is used to select the best substitute node. The specific process is shown in Algorithm 8.

Algorithm 8 NI-MOD-P-RRT*

Input: $X_{start}, X_{goal}, X_{obs}, Maxiter, R_{near}$

Output: $G=(V,E)$, $T=(V,C_{new})$

1: $V \leftarrow \{X_{goal}\}; E \leftarrow \varnothing;$

2:**for** $i=1$ **to** $Maxiter$ **do**
3: **if** δ<p **then**
4: $X_{rnd} = random(X_{free})$;
5: **else**

$X_{rnd} = X_{start}$;
end if
$X_{prnd} \leftarrow RGD\left(X_{rnd}, d_{obs}^{*}, \lambda\right)$;
$X_{nearest} \leftarrow Nearst\left(V, X_{prnd}\right)$;
$\theta = angle(X_{goal}, X_{start})$;
$\beta = angle(X_{prnd}, X_{nearest})$;
if $\left(\theta + \frac{\pi}{2}\right) > \beta > \left(\frac{\pi}{2} - \theta\right)$;
$X_{new} = Steer\left(X_{nearest}, X_{prnd}\right)$;
if $noCollision\left(X_{nearest}, X_{new}, X_{obs}\right)$ **then**
$X_{nears} \leftarrow Near\left(V, X_{new}, R_{near}\right)$;
$X_{parent} \leftarrow ChooseParent\left(X_{nears}, X_{new}, X_{nearest}\right)$;
$C_{new} \leftarrow \mathrm{Cos}t\left(X_{parent}\right) + Dist\left(X_{new}, X_{parent}\right)$;
$V \leftarrow V \cup \left\{X_{new}\right\}$;
$E = E \cup \left\{\left(X_{parent}, X_{new}\right)\right\}$;
$G \leftarrow \mathrm{Re}wire\left(G, X_{new}, X_{nears}\right)$;
$T = T \cup \left(V, C_{new}\right)$;
if $noCollision\left(X_{new}, X_{start}\right)$ **then**
return $G = \left(V, E\right)$;
$G \leftarrow$ ITTI Rewire$\left(V, E\right)$;
$G \leftarrow$ B-SS Rewire$\left(V, E\right)$
end if
end if
end if;
end for
Return $G = \left(V, E\right)$;
navigation

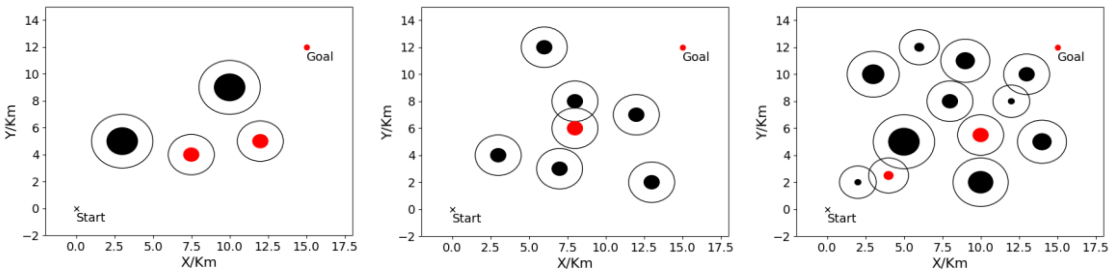
ITTI: Uses improved triangle inequality for pruning;
B-SS: Uses B-spline for smoothing.

4. Simulation experiments and analysis

4.1. Experimental environment

The PyCharm simulation software is used to conduct experiments on the NI-MOD-P-RRT* algorithm proposed in this chapter, as well as the RRT*, P-RRT*, and Informed-RRT* algorithms, to verify the effectiveness of the NI-MOD-P-RRT* algorithm in the path planning phase. Dynamic environment simulations are also performed to validate the feasibility of the NI-MOD-P-RRT* algorithm in dynamic environments. To ensure the rigor of the experiments, the parameters of the

algorithms are kept consistent. All map environments are set on a 20 × 17 two-dimensional plane, with 200 iterations per experiment. Each map is tested 30 times, and the average values of each metric are taken as the experimental results. The three experimental environments are shown in Figures 4-10. The results reveal that Environment (A) simulates the addition of aquaculture equipment in an aquaculture setting, and Environments (B) and (C) simulate scenarios where the positions of aquaculture equipment shift due to factors such as wind and waves. The search starts at (0,0), and the target point is (15,12). Black areas represent known obstacle regions, red areas denote unknown obstacle regions, circles indicate obstacle envelopes, green line segments represent random trees generated during the search, and bold red line segments show the final path generated by the random tree.



Environment (A) **Environment (B)** **Environment (C)**

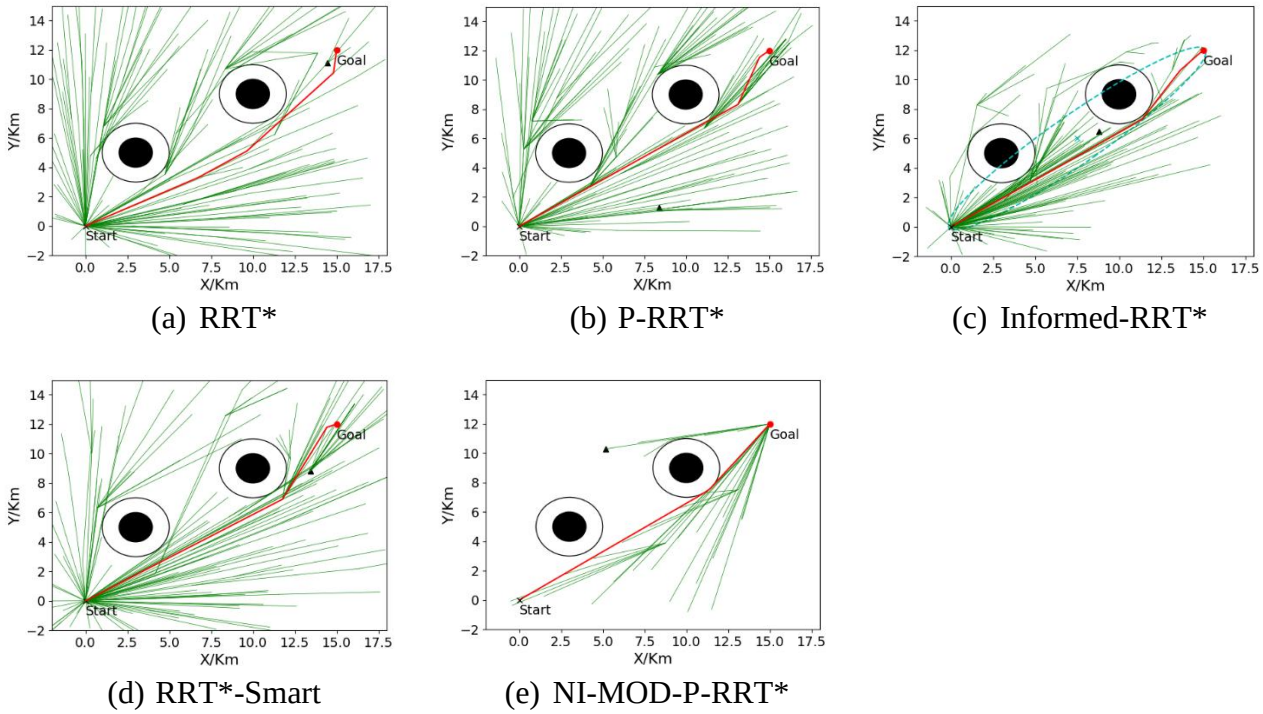
Figure 11. Experimental environment.

4.2. Simulation results in static environments

First, the NI-MOD-P-RRT* algorithm, RRT*, P-RRT*, Informed-RRT*, and RRT*-Smart algorithms are tested separately in the experimental environment with red obstacles removed in Figure 11. By comparing and analyzing the path length, convergence time, and iteration count of the four algorithms, the effectiveness of the proposed algorithm in the path planning phase is validated.

4.2.1. (A) Environment

The results of the five algorithms in environment (A) are shown in Figure 12, where the feasible trajectory of the unmanned ship is indicated in red. In Figure 12 (a), since RRT* adopts random sampling in space, more invalid nodes appear in the path, and the average path is the longest. Figure 12 (b) shows the simulation diagram of P-RRT*. Compared with RRT*, due to the introduction of the artificial potential field method, the reverse search in its search is significantly lessened, and the nodes tend toward the end point, so the path is better. In Figure 12 (c), the path is made shorter by limiting the sampling range. In Figure 12 (d), the RRT*-Smart algorithm is optimized through intelligent sampling, and the sampling points are concentrated near the inflection points. In the environment in Figure 12 (e), the proposed NI-MOD-P-RRT* algorithm has obvious advantages, with a shorter path and fewer nodes, which can meet the performance requirements of unmanned ships.

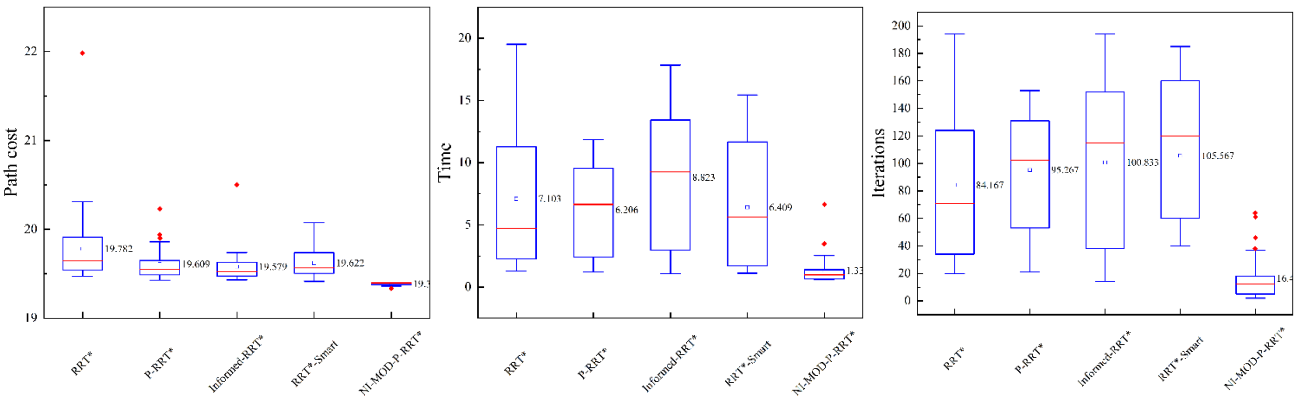


(a) RRT* (b) P-RRT* (c) Informed-RRT*

(d) RRT*-Smart (e) NI-MOD-P-RRT*

Figure 12. Simulation in environment (A).

The simulation data of the initial path calculated by each algorithm in environment (A) includes the average running time, average path length, and average number of iterations of 30 experiments, as shown in Figure 13.



(a) Path cost (b) Time (c) Iterations

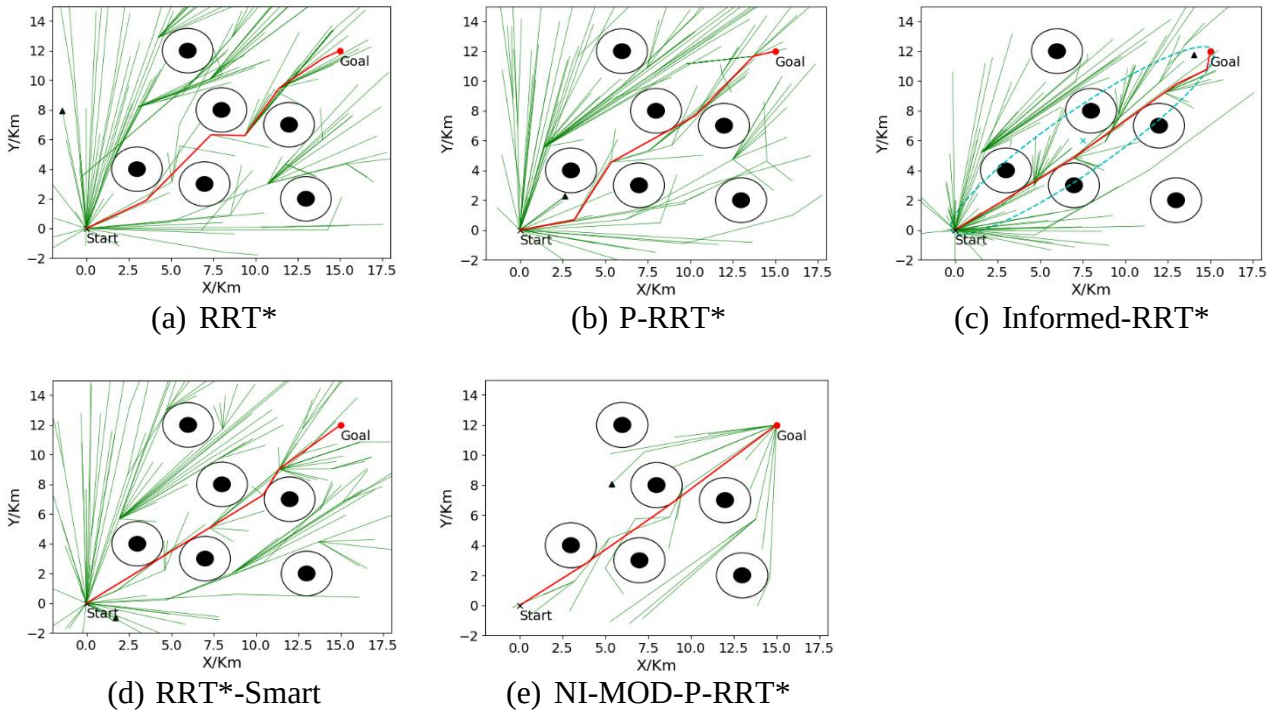
Figure 13. Simulation data in environment (A).

In Table 1, the average length of the 30 experimental data of the algorithm proposed in this paper is 19.3865 kilometers, the average running time is 1.3306 seconds, and the average number of iterations is 16. Compared with the other four algorithms, the average length of the generated trajectories decreased by 2.0%, 1.14%, 0.98%, and 1.2% respectively, the average running time decreased by 81%, 79%, 85%, and 79% respectively, and the average number of iterations decreased by 81%, 83%, 84%, and 85% respectively. In this paper, the average path length of the improved

algorithm does not decrease much. The main reason is that the RRT* series of algorithms have the characteristic of progressive optimality, and the working environment map adopted in this paper is relatively small. However, it can be seen from Figure 13 that the path length of each simulation of the improved algorithm does not vary much and has high stability. The experimental results show that this algorithm plans a path with lower cost in a simple environment, and the number of algorithm iterations and running time are effectively reduced.

4.2.2. (B) Environment

The results of the five algorithms in Environment (B) are shown in Figure 14. Figure 14(e) demonstrates that the trajectory planned by the proposed algorithm in Environment (B) has almost no inflection points, indicating superior performance.

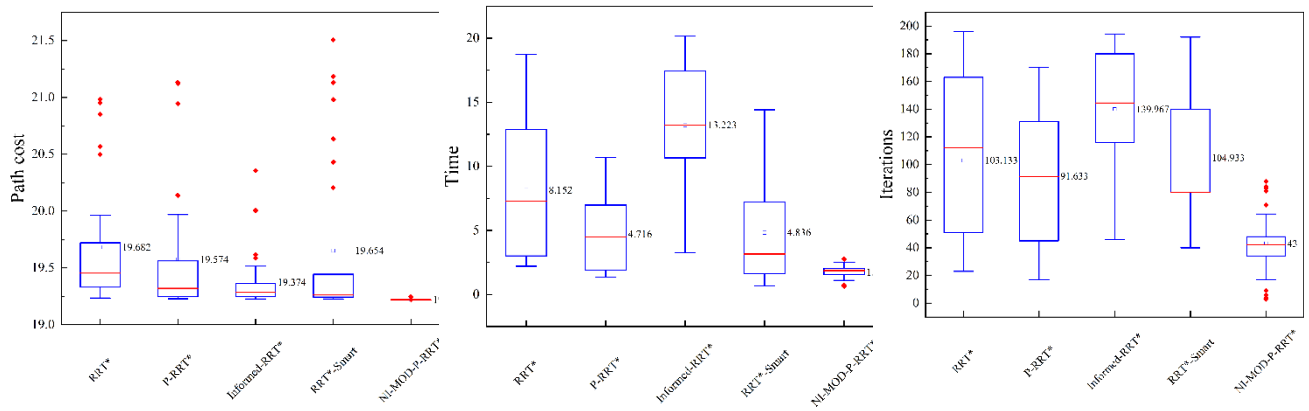


(a) RRT* (b) P-RRT* (c) Informed-RRT*

(d) RRT*-Smart (e) NI-MOD-P-RRT*

Figure 14. Simulation in environment (B).

Analysis of Figure 15 shows that compared to other algorithms, the proposed algorithm significantly improves all of three performance metrics. In Table 1, the average path length of this algorithm is 19.2210 kilometers, the average runtime is 1.7463 seconds, and the average iteration count is 43. Compared to the other four algorithms, the average runtime is reduced by 79%, 63%, 87%, and 64%, respectively, and the average iteration count is reduced by 58%, 53%, 69%, and 59%. The comparative data confirms that this algorithm achieves better path planning.

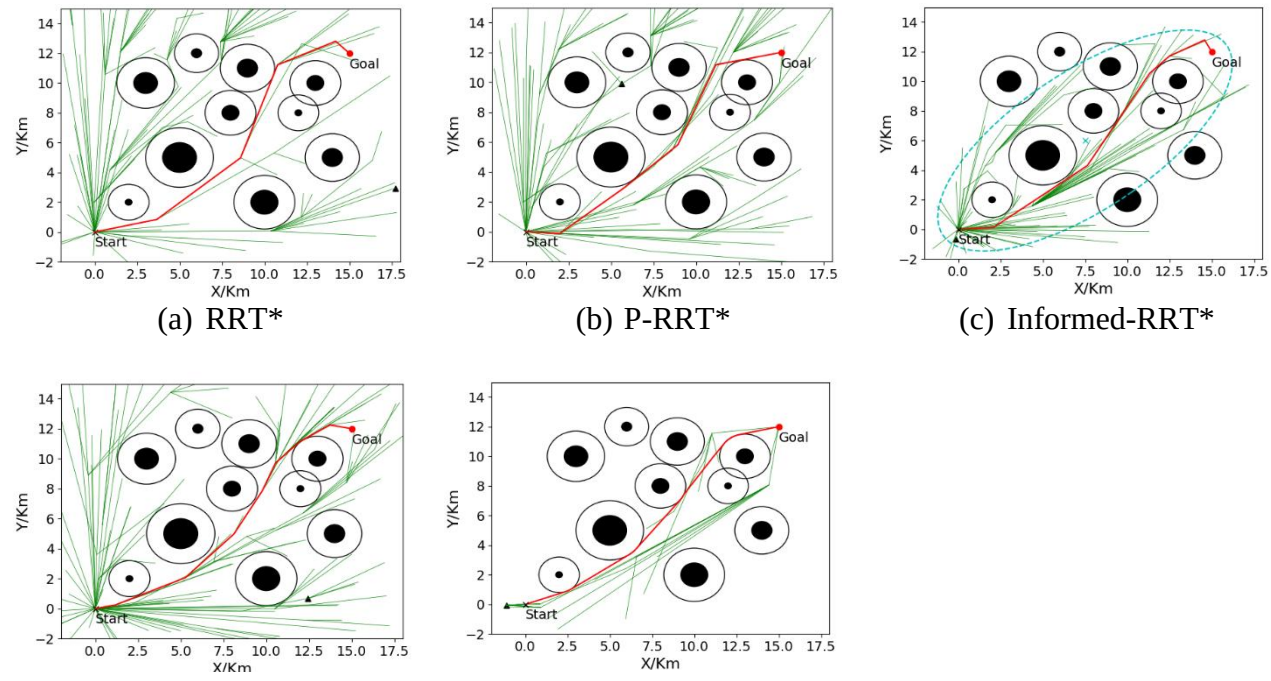


(a) Path cost (b) Time (c) Iterations

Figure 15. Simulation data in environment (B).

4.2.3. (C) Environment

Compared to Environments (A) and (B), Environment (C) features numerous obstacles of varying sizes, posing challenges for unmanned vessel trajectory planning. However, as shown in Figure 16(e), the improved algorithm proposed in this chapter efficiently plans paths in this complex environment, generating fewer redundant points and smoother trajectories.



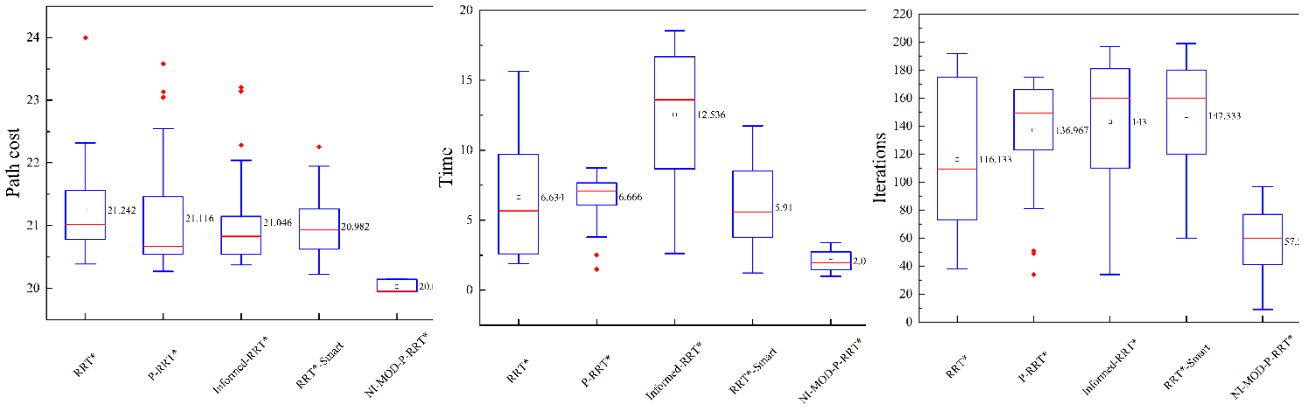
(a) RRT* (b) P-RRT* (c) Informed-RRT*

(d) RRT*-Smart (e) NI-MOD-P-RRT*

Figure 16. Simulation in environment (C).

Analysis of Table 1 reveals that the proposed algorithm achieves an average path length of 20.0245 kilometers, an average runtime of 2.0774 seconds, and an average iteration count of 58 for initial solutions in 30 experiments in Environment (C). Compared to the other four algorithms, the average runtime is reduced by 69%, 69%, 83%, and 65%, respectively, and the average iteration

count is reduced by 50%, 57%, 59%, and 61%. Despite incorporating pruning and smoothing, the optimized sampling angles and connection conditions in the improved algorithm lessen the average runtime while enhancing performance metrics, as evidenced by Figure 17. The results demonstrate the superiority of the proposed algorithm even in complex environments.



(a) Path cost (b) Time (c) Iterations

Figure 17. Simulation data in environment (C).

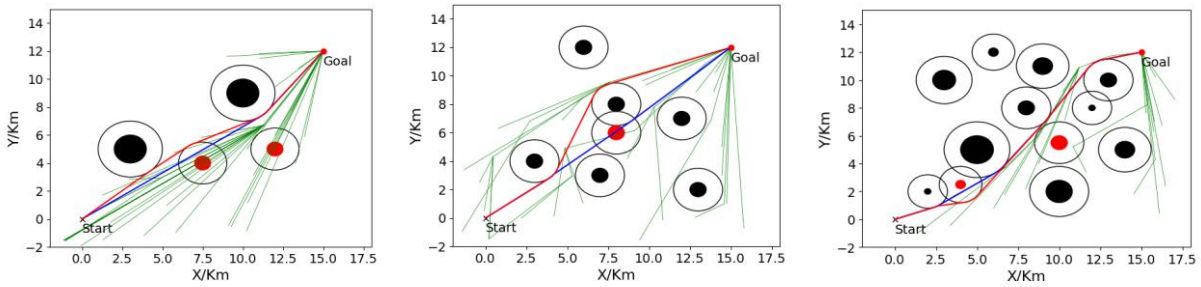
Table 1. Average simulated data in the three environments

Environment	Algorithm	Average path cost/km	Average time/s	Average iterations
(A)	RRT*	19.7815	7.1030	84
	P-RRT*	19.6092	6.2061	95
	IN-RRT*	19.5789	8.8230	101
	RRT*-Smart	19.6221	6.4087	106
	NI-MOD-P-RRT*	**19.3865**	**1.3306**	**16**
(B)	RRT*	19.6824	8.1523	103
	P-RRT*	19.5740	4.7155	92
	IN-RRT*	19.3741	13.2230	140
	RRT*-Smart	19.6540	4.8362	105
	NI-MOD-P-RRT*	**19.2210**	**1.7463**	**43**
(C)	RRT*	21.2420	6.6344	116
	P-RRT*	21.1163	6.6662	137
	IN-RRT*	21.0459	12.5359	143
	RRT*-Smart	20.9821	5.9097	147
	NI-MOD-P-RRT*	**20.0245**	**2.0774**	**58**

Summary: Table 1 shows that the proposed NI-MOD-P-RRT* algorithm significantly outperforms the other three algorithms in average runtime and iteration count. Compared to P-RRT*, the addition of sampling angle constraints and optimized conditions accelerates convergence. Although the average path length improvement is marginal, the trajectories in Figures 12(d), 14(d), and 16(d) demonstrate that the proposed algorithm generates smoother paths, which are better suited for unmanned vessel motion patterns.

4.3. Simulation results in dynamic environments

The environment for dynamic path planning simulation employs the three scenarios illustrated in Figure 11. Known obstacles are marked in black, while unknown obstacles are indicated in red. Figure 18 depicts the initial paths (blue), actual trajectories during navigation (red), and reconnected tree structures across these three environments. It is evident that the initial paths in all scenarios would collide with these unknown obstacles, and the path replanning strategy successfully assists the unmanned vessel in avoiding them. Table 2 summarizes the path replanning outcomes for the three environments, including the number of candidate nodes, optimal node count from the greedy algorithm, coordinates of optimal nodes, and replanning duration.



（a）A environments　　（b）B environments　　（c）C environments

Figure 18. Replanning results under three environments.

Table 2. Path replanning data in the three environments.

Environments	Candidate	Greed-optimal	Best node	Time(ms)
(A)	31	2	(6.93，5.39)	40
(B)	8	2	(5.84，8.45)	30
(C)	25	1	(4.53，1.31)	14

Based on these experimental results, the observations are as follows:

(1) The results indicate that the NI-MOD-P-RRT* algorithm can generate feasible paths in all three environments, effectively helping unmanned surface vessels navigate around known and unknown obstacles. This demonstrates that the path replanning scheme can successfully select optimal alternative nodes, proving the feasibility of the approach.

(2) The number of candidate nodes varies across different environments, ranging from a few to dozens, which affects execution time. However, the replanning execution time is consistently less than 100ms, reflecting the efficiency of the path replanning scheme.

5. Conclusions

We address the path planning problem for USV in dynamic environments by proposing the NI-MOD-P-RRT* trajectory planning algorithm, which enables dynamic environment replanning based on P-RRT*. Simulation experiments in static environments with various maps show that improvements to the sampling strategy and connection constraints of the P-RRT* algorithm

significantly lessen its runtime and iteration count. Pruning and smoothing the initial path brings it closer to obstacle edges, shortening the path length and making it more suitable for USV navigation. Additionally, the proposed algorithm stores node information as prior knowledge, enabling the path replanner to select alternative tree nodes for collision avoidance when unknown obstacles block the current path. Dynamic environment simulations demonstrate that NI-MOD-P-RRT* can select replacement nodes within 0.1 seconds during replanning, validating its effectiveness in simulated scenarios.

we focus solely on trajectory planning algorithms for static unknown obstacles. The algorithm in this paper also has some limitations: The limitation of sampling angles may lead to local optimal solutions; the performance of the time complexity of dynamic re-programming in extreme scenarios (such as dense obstacles); and the consumption characteristics of computing resources by target bias sampling. Trajectory planning for USV in environments with moving unknown obstacles and real-world experiments will be our focus in future research.

Author contributions

Yunlai Fu conducted simulations, improved the algorithm, and wrote the first edition of the manuscript. Wei Wang conducted a strict review of the technical and scientific content of the manuscript and helped improve the overall structure and clarity of the paper. Yongqi Zhang sorted out the current research status at home and abroad and assisted in analyzing the results. Yan Liu and Haoran Li participated in the analysis and interpretation of the results. All authors reviewed and approved the final version of the manuscript

Use of Generative-AI tools declaration

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

Acknowledgments

This work was supported in part by the General Scientific Research Projects of Liaoning Province Science and Technology Joint Plan Project under Grant 2024JH2/102600083, and in part by Liaoning Provincial Department of Education under Grant JYTMS20230489.

Conflict of interest

All authors declare no conflicts of interest in this paper.

References

1. Wu Y, Duan Y, Wei Y, An D, Liu J (2022) Application of intelligent and unmanned equipment in aquaculture: A review. *Comput Electron Agr* 199: 107201. https://doi.org/10.1016/j.compag.2022.107201

2. Jones M, Djahel S, Welsh K (2023) Path-planning for unmanned aerial vehicles with environment complexity considerations: A survey. *ACM Comput Surv* 55: 1–39. https://doi.org/10.1145/3570723
3. Sang H, You Y, Sun X, Zhou Y, Liu F (2021) The hybrid path planning algorithm based on improved A* and artificial potential field for unmanned surface vehicle formations. *Ocean Eng* 223: 108709. https://doi.org/10.1016/j.oceaneng.2021.108709
4. Yu J, Yang M, Zhao Z, Wang X, Bai Y, Wu J, et al. (2022) Path planning of unmanned surface vessel in an unknown environment based on improved D* Lite algorithm. *Ocean Eng* 266: 112873. https://doi. org/10.1016/j.oceaneng.2022.112873
5. Miao C, Chen G, Yan C, Wu Y (2021) Path planning optimization of indoor mobile robot based on adaptive ant colony algorithm. *Comput Ind Eng* 156: 107230. https://doi.org/10.1016/j.cie.2021.107230
6. Wang X, Liu Z, Li X (2023) Optimal delivery route planning for a fleet of heterogeneous drones: A rescheduling-based genetic algorithm approach. *Comput Ind Eng* 179: 109179. https://doi.org/10.1016/j.cie.2023.109179
7. Gad AG (2022) Particle swarm optimization algorithm and its applications: a systematic review. *Arch comput method eng* 29: 2531–2561. https://doi.org/10.1007/s11831-021-09694-4
8. Charalambopoulos N, Xidias E, Nearchou A (2023) Efficient ship weather routing using probabilistic roadmaps. *Ocean Eng* 273: 114031. https://doi.org/10.1016/j.oceaneng.2023.114031
9. LaValle SM, Kuffner Jr, JJ (2001) Randomized kinodynamic planning. *The international journal of robotics research* 20: 378–400. http://dx.doi.org/10.1177/02783640122067453
10. Billard A, Kragic D (2019) Trends and challenges in robot manipulation. *Science* 364: eaat8414. https://doi.org/10.1126/science.aat8414
11. Ganesan S, Ramalingam B, Mohan RE (2024) A hybrid sampling-based RRT* path planning algorithm for autonomous mobile robot navigation. *Expert Syst Appl* 258: 125206. https://doi.org/10.1016/j.eswa.2024.125206
12. Yang Q, Qu W, Wang Y, Song X, Guo Y, Ke Y (2025) Smooth joint motion planning for redundant fiber placement manipulator based on improved RRT. *Robot Comput-Int Manuf* 91: 102851. https://doi.org/10.1016/j.rcim.2024.102851
13. Fan J, Chen X, Liang X (2023) UAV trajectory planning based on bi-directional APF-RRT* algorithm with goal-biased. *Expert syst appl* 213: 119137. https://doi.org/10.1016/j.eswa.2022.119137
14. Liu Y, Li C, Yu H, Song C (2023) NT-ARS-RRT: A novel non-threshold adaptive region sampling RRT algorithm for path planning. *J King Saud Univ-Comput Inform Sci* 35: 101753. https://doi.org/10.1016/j.jksuci.2023.101753
15. Gu Q, Zhen R, Liu J, Li C (2023) An improved RRT algorithm based on prior AIS information and DP compression for ship path planning. *Ocean Eng* 279: 114595. https://doi.org/10.1016/j.oceaneng.2023.114595
16. Zhou Y, Zhang E, Guo H, Fang Y, Li H (2021) Lifting path planning of mobile cranes based on an improved RRT algorithm. *Adv Eng Inform* 50: 101376. https://doi.org/10.1016/j.aei.2021.101376

17. Karaman S, Emilio F (2011) Sampling-based algorithms for optimal motion planning. *The international journal of robotics research* 30: 846–894. http://dx.doi.org/10.1177/0278364911406761
18. Gammell JD, Srinivasa SS, Barfoot TD (2014) Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ international conference on intelligent robots and systems*, 2997–3004. IEEE. https://doi.org/10.1109/IROS.2014.6942976
19. Islam F, Nasir J, Malik U, Ayaz Y, Hasan O (2012) Rrt∗-smart: Rapid convergence implementation of rrt∗ towards optimal solution. In *2012 IEEE international conference on mechatronics and automation*, 1651–1656. IEEE.
20. Jeong IB, Lee SJ, Kim JH (2019) Quick-RRT*: Triangular inequality-based implementation of RRT* with improved initial solution and convergence rate. *Expert Syst Appl* 123: 82–90. https://doi.org/10.1016/j.eswa.2019.01.032
21. Qureshi AH, Ayaz Y (2016) Potential functions based sampling heuristic for optimal path planning. *Auton Robot* 40: 1079–1093. https://doi.org/10.1007/s10514-015-9518-0
22. Wang Z, Li P, Wang Z, Li Z (2024) APG-RRT: Sampling-based path planning method for small autonomous vehicle in closed scenarios. *IEEE Access* 12: 25731–25739. https://doi.org/10.1109/ACCESS.2024.3359643
23. Liang YM, Zhao HY (2023) CCPF-RRT*: An improved path planning algorithm with consideration of congestion. *Expert Syst Appl* 228: 120403. https://doi.org/10.1016/j.eswa.2023.120403
24. Yu S, Chen J, Liu G, Tong X, Sun Y (2023) SOF-RRT*: An improved path planning algorithm using spatial offset sampling. *Eng Appl Artif Intell* 126: 106875. https://doi.org/10.1016/j.engappai.2023.106875
25. Yang F, Liu J, Li S, Hu X (2025) Minimum jerk motion planning for maritime autonomous surface ships based on Bézier curves. *J Mar Eng Technol* 24: 85–97. https://doi.org/10.1080/20464177.2024.2434331
26. Reda M, Onsy A, Haikal AY, Ghanbari A (2025) A novel reinforcement learning-based multi-operator differential evolution with cubic spline for the path planning problem. *Artif Intell Rev* 58: 142. https://doi.org/10.1007/s10462-025-11129-6
27. Huo F, Zhu S, Dong H, Ren W (2024) A new approach to smooth path planning of Ackerman mobile robot based on improved ACO algorithm and B-spline curve. *Robot Auton Syst* 175: 104655. https://doi.org/10.1016/j.robot.2024.104655
28. Zhao F, Zhuang C, Wang L, Dong C (2024) An iterative greedy algorithm with Q-learning mechanism for the multiobjective distributed no-idle permutation flowshop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 54: 3207–3219. https://doi.org/10.1109/TSMC.2024.3358383

AIMS Press