

Article

An Integrated Path Planning Algorithm Based on Vector Jump Point Search and Ant Colony Optimization for Quay Crane Inspection

Haoye Zhang ¹, Zihan Lei ¹, Mingxiao Wang ¹, Zhipeng Hou ¹, Hongren Zhao ¹, Christophe Claramunt ², Gang Tang ^{1,*}  and Weidong Zhu ^{3,*} 

¹ Logistics Engineering College, Shanghai Maritime University, Shanghai 201306, China

² Naval Academy Research Institute, F-29240 Lanvéoc, France

³ Department of Mechanical Engineering, University of Maryland Baltimore Country, Baltimore, MD 21250, USA

* Correspondence: gangtang@shmtu.edu.cn (G.T.); wzhu@umbc.edu (W.Z.)

Abstract

Unmanned aerial vehicles have become a promising platform for inspecting large port machinery; however, quay cranes contain sparse but complex steel-frame structures, multiple inspection points, and narrow collision-constrained spaces, which make efficient inspection path planning difficult. Existing approaches often focus either on global point sequencing or local collision-free search, and conventional global optimizers usually use straight-line distances that do not reflect obstacle-constrained flight costs. This paper proposes an integrated path planning method for quay crane inspection based on vector jump point search and ant colony optimization. In the local path-searching stage, vector-guided preprocessing and path simplification are used to calculate collision-free paths between mission points and construct a path cost matrix. In the global optimization stage, ant colony optimization determines the inspection sequence using the collision-free cost matrix rather than Euclidean distances. Simulation experiments were conducted on a simplified quay crane model of 132 m × 22 m × 70 m with 25 mission points. The results show that the proposed method reduced the average local path-searching time from 5.3392 s to 4.2907 s, corresponding to a 19.6% improvement over jump point search, while reducing the average local path length by 5.1%. The final global inspection path obtained in the experimental case was 516.1 m, which was shorter than those obtained by simulated annealing, genetic algorithm, particle swarm optimization, and the previous method. These results indicate that the proposed method can improve local planning efficiency and provide an effective inspection route for unmanned aerial vehicle-based quay crane inspection.

Keywords: vector jump point search; ant colony optimization; path planning; unmanned aerial vehicle; quay crane inspection



Academic Editor: Philip Moore

Received: 20 May 2026

Revised: 29 June 2026

Accepted: 30 June 2026

Published: 3 July 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

As a part of port transportation, a quay crane plays a crucial role in achieving efficient transportation of goods between land and sea [1]. Due to frequent operations, the quay crane's health test is extremely paramount for the normal operation of the port [2]. Usually, the quay crane's health test is carried out by specialized personnel, which is time-consuming and laborious work. This not only requires frequent long-time shutdown of the equipment but also generates hidden danger to the workers and decreases the economic efficiency [3].

Recently, because of the low cost and operation risk, an unmanned aerial vehicle (UAV) has been widely used in the detection and inspection tasks [4–6]. In order to quickly monitor the quay crane condition, the UAV needs to plan an obstacle-free inspection path with minimal length while ensuring passing through all the mission points.

The key difficulty in UAV-based quay crane inspection is that the inspection route must satisfy two requirements simultaneously: it must avoid the three-dimensional steel structure locally, and it must visit multiple vulnerable components globally with limited flight distance and time. Traditional global optimization algorithms are effective for sequencing discrete points, but they often rely on Euclidean distances and therefore cannot directly represent collision-free flight costs in obstacle-rich crane structures. Conversely, local graph-search algorithms can generate feasible paths between two points, but repeatedly applying them to all mission-point pairs may increase computation time in a large three-dimensional grid.

To address this problem, this study integrates vector jump point search with ant colony optimization. The vector jump point search component accelerates local collision-free path searching by pruning geometrically inefficient initial directions and simplifying unnecessary turning points. The ant colony optimization component then optimizes the inspection sequence based on the collision-free path cost matrix generated by the local planner. Thus, the proposed framework is not a simple parallel use of two algorithms; rather, the local planner provides physically meaningful edge costs for the global optimizer.

The main contributions of this study are as follows: (1) an integrated local–global path-planning framework is developed for UAV-based quay crane inspection; (2) a vector-guided local search strategy is used to reduce unnecessary node expansion in a structured crane environment; (3) the global inspection sequence is optimized using collision-free costs rather than straight-line distances; and (4) simulation results are reported with explicit performance metrics, including path-searching time, path length, expanded nodes, turning points, and global route length.

The remainder of this paper is organized as follows. Section 2 reviews related research and summarizes the technical background. Section 3 formulates the quay crane inspection path-planning problem. Section 4 presents the proposed integrated method. Section 5 describes the evaluation and experimental settings. Section 6 reports and analyzes the simulation results. Section 7 discusses the findings, limitations, and engineering implications. Section 8 concludes the paper.

2. Related Research

2.1. UAV Inspection and Mission-Point Generation

The first step of the UAV inspection task is to determine the suitable inspection mission points. Freimuth and König [7] proposed a method to generate mission points for the desired location by considering different camera view angles. Englot and Hover [8] introduced a sampling method to select the mission points for a ship hull. Tan et al. [9] used environment information and building information modeling (BIM) model to generate mission points of building surface inspection. Overall, for different inspection missions, reasonable inspection mission points should be selected to meet the requirements of the specific task.

2.2. Global Inspection-Sequence Optimization

After determining all the mission points of UAV, the second step of the UAV inspection task is to determine an appropriate inspection sequence. So far, most researchers treat the problem as a traveling salesman problem (TSP). This leads to a nondeterministic polynomial (NP) problem for finding the shortest path of a salesman who needs to visit every

point once and return to the initial point [10,11]. Fischetti and Toth developed additive bounding procedures for the asymmetric traveling salesman problem and embedded them in a branch-and-bound framework [12], but the computational time remains expensive as the problem scale increases. To accelerate the computing speed of the TSP, several intelligent optimization algorithms have been proposed, such as simulated annealing (SA) [13], genetic algorithm (GA) [14], particle swarm optimization (PSO) [15], and ant colony optimization (ACO) [16]. Among these algorithms, the ACO algorithm has the characteristics of distributed computation, positive feedback, and heuristic search. In essence, it is a heuristic global optimization algorithm with the interesting characteristics of parallelism, robustness, and self-organization, it only relies on the ability of the population to search for the optimal solution [17–19]. However, common ACO algorithm solves the TSP by regarding the path distance as the straight-line distance between two mission points, which is not suitable in an environment with obstacles.

2.3. Local Collision-Free Path Searching

To avoid UAV collision, the third step of UAV inspection task is to calculate a collision-free path between two points. Generally, there are two types of collision-free path planning algorithms: randomized search algorithms and graph search algorithms. Randomized search algorithms include probabilistic roadmap (PRM) [20], rapidly exploring random tree (RRT) [21], and optimal rapidly exploring random tree (RRT*) [22]. These methods can find a collision-free path between two points, but their random behavior makes the algorithm's performance unpredictable when we need to re-plan the path. Graph search algorithms include Dijkstra [23], A-star (A*) [24], and sparse A-star search (SAS) [25]; these algorithms can plan the shortest collision-free path between two points. Therefore, the graph search algorithm and intelligent optimization algorithm can be combined to plan the shortest inspection path through all the mission points. In [26], a path planning method combining A* algorithm and GA for bridge inspection was proposed to find an obstacle-free path with minimum length. Huang et al. [27] introduced an electric power inspection path planning based on the improved ACO and A* algorithm. They used A* for the local path planning and ACO for the global path planning, and the simulation proved that the energy consumption and time consumption of the improved ACO-A* algorithm are less than that of the A* algorithm. However, in a large-scale environment, A* would consume a lot of time to find the optimal collision-free path between two points [28,29], which will increase a lot of time to generate the path cost matrix for the global path planning. The jump point search (JPS) solves the above problem by planning in uniform-cost grid map, which dramatically increases the searching speed compared with the A* algorithm [30]. Liu et al. [31] extended the JPS algorithm to three-dimensional environments, proving that the search speed of the JPS algorithm is also better than the A* algorithm in three-dimensional environments. However, the standard JPS algorithm is designed for general uniform grid maps and does not fully exploit the structural characteristics of the quay crane environment. Unlike maze-like indoor environments, a quay crane is a large-scale skeleton structure characterized by sparse obstacles and semi-open spaces. The proposed VJPS differs from existing JPS variants by introducing a vector-based preprocessing rule specifically designed for such structures. By utilizing the start-goal vector guidance, VJPS aggressively prunes search directions that are geometrically unlikely to yield optimal paths in this specific domain, thereby significantly reducing the search space designated for large-scale industrial inspection. Recent studies have also emphasized safety-aware and task-specific UAV trajectory planning, including contingency-aware urban UAV planning [32] and maritime search-and-rescue route planning under moving-target conditions [1].

2.4. Technical Background and Research Gap

From a technical perspective, UAV inspection path planning in a quay crane environment can be regarded as a two-level problem. At the lower level, the planner must generate a collision-free path between two inspection points in a three-dimensional grid map. At the upper level, it must determine the visiting sequence of all inspection points so that the total inspection route is shortened. A key limitation of many existing studies is that local obstacle avoidance and global point sequencing are handled separately or that the global optimizer uses straight-line distances that ignore the actual obstacle-constrained flight cost. The research gap addressed in this paper is therefore the integration of fast local collision-free cost calculation with global inspection-sequence optimization for structured quay crane inspection scenarios.

3. Problem Statement

In a quay crane environment, most of the above studies cannot quickly plan a collision-free inspection path to inspect vulnerable areas of the quay crane. Our paper takes two steps to solve the problem: namely local path planning and global path planning, to find a short and collision-free inspection route passing through all the mission points of a quay crane. In the local path planning phase, a collision-free path between every two mission points can be quickly found using the VJPS algorithm, and a path cost table is thus built. In the global path planning phase, the ACO algorithm is applied to obtain an optimized global inspection sequence based on this obstacle-aware cost information.

Compared with our previous work in [33], this study further improves the local path-planning speed and shortens the UAV inspection path by determining a reasonable inspection sequence. The present manuscript also provides a clearer problem formulation, a more explicit integration mechanism between local and global planning, and a more detailed analysis of experimental limitations and engineering applicability.

Problem Description and Mathematical Formulation

A port quay crane is a complex system. Due to the frequent operations, quite a few parts are prone to be damaged. To avoid the failure and unnecessary economic loss, a quay crane needs periodic inspection. This paper mainly focuses on the UAV inspection for the quay crane steel structure. The quay crane steel structure consists of door frames, horizontal beams, a pylon frame system, a stay system, etc. The crane steel structure's life prediction can be regarded as the process of the initial crack with a certain length from the propagation to fracture. The initial crack is obtained by actual inspection, and it is also the data needed to make the detection with the UAV. According to the inspectors' experience, the steel structure crack usually occurs in the location of fatigue source and stress concentrations, which is the crucial inspection area of the UAV.

The simulation model of the quay crane is shown in Figure 1, in which the length, width, and height are 132 m, 22 m, and 70 m respectively, and the possible mission points of the UAV are represented by yellow stars. Due to the quay crane's overall structure's symmetry, we use F and B to represent the quay crane's front side and back side, respectively. As shown in Table 1, there are totally 25 mission points, and point 1 represents the first mission point of the UAV. Considering the limited power, the UAV needs to plan the shortest obstacle-free path, which starts from point 1, passes through all the mission points to carry out the inspection tasks (such as taking photos of vulnerable positions), and finally returns the initial point (point 1).

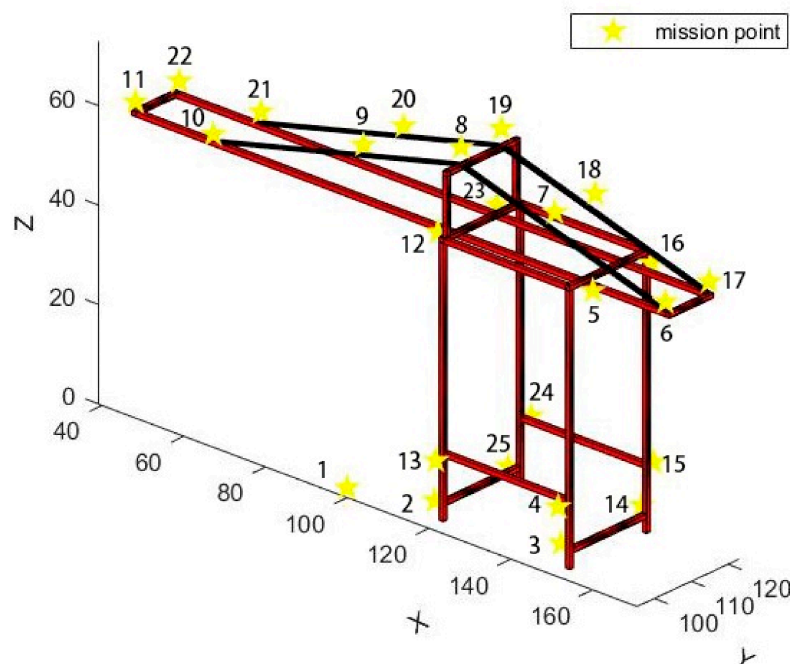


Figure 1. Simplified model of a quay crane and its mission points.

Table 1. Mission points to carry out the task.

F	1	2	3	4	5	6	7	8	9	10	11	12	13
B	14	15	16	17	18	19	20	21	22	23	24	25	

To generate the shortest collision-free path passing through all the mission points, the first task is to quickly plan an obstacle-free path between every two mission points, and the second task is to determine the proper inspection sequence of the UAV.

By defining a binary variable:

$$x_{ij} = \begin{cases} 1 & \text{the path goes from point } i \text{ to point } j \\ 0 & \text{otherwise} \end{cases}, \quad (1)$$

The UAV inspection path-planning problem can be formulated as follows:

$$\min c_{ij} \quad i = 1, \dots, n \quad j = 1, \dots, n, \quad (2)$$

$$\min \sum_{i=1}^n \sum_{j \neq i, j=1}^n c_{ij} x_{ij}, \quad (3)$$

$$s.t. \sum_{i=1, i \neq j}^n x_{ij} = 1 \quad j = 1, \dots, n, \quad (4)$$

$$\sum_{j=1, j \neq i}^n x_{ij} = 1 \quad i = 1, \dots, n, \quad (5)$$

$$\sum_{i \in T} \sum_{j \neq i, j \in T} x_{ij} \leq |T| - 1 \quad \forall T \subsetneq \{1, \dots, n\}, |T| \geq 2, \quad (6)$$

where n represents the number of the UAV's mission points, c_{ij} is the path cost from point i to point j , and this paper takes the collision-free path length as the path cost. This paper divides the inspection path-planning problem into two sub problems: local path planning and global path planning, and there are two objective functions. Equation (2) is the objective function of the local path-planning problem; this optimization objective is to minimize the local path length between every two mission points. Equation (3) is the objective function of the global path-planning problem; this optimization objective is to minimize the global path length through all mission points. The constraint in Equation (4) implies

that each mission point is derived from exactly one other mission point, and the constraint in Equation (5) means that from each mission point there is a departure to exactly one other mission point. Condition (6) ensures no sub-tours, i.e., the UAV returns to the initial mission point after visiting all the other mission points, T is the set of all the mission points, and $|T|$ denotes the number of mission points that the UAV passes through.

In this study, to focus on the geometric optimality of the inspection path, the UAV is modeled as a kinematic particle (point mass). We assume the UAV operates at low speeds during the inspection task, allowing the low-level flight controller to effectively handle dynamic constraints and minor environmental disturbances (such as wind gusts) to track the generated trajectory. While this kinematic model simplifies the planning process, it does not explicitly account for the UAV's dynamic constraints (e.g., maximum acceleration, turning radius) or complex aerodynamic effects. Future work will integrate these dynamic constraints into a trajectory smoothing phase.

4. Materials and Methods

The overall workflow of the proposed hybrid framework is shown in Figure 2. The method starts with the quay crane grid model and predefined inspection mission points. VJPS is applied as the local planner to calculate collision-free paths between mission-point pairs and to construct an obstacle-aware path cost matrix. The cost matrix is then passed to ACO as edge-cost information for global sequence optimization, and the final route is obtained by concatenating the selected VJPS path segments according to the ACO-derived inspection sequence. The dashed box in Figure 2 denotes the coupling layer, i.e., the intermediate mechanism that transforms the VJPS-generated local path costs into the obstacle-aware cost matrix used by ACO.

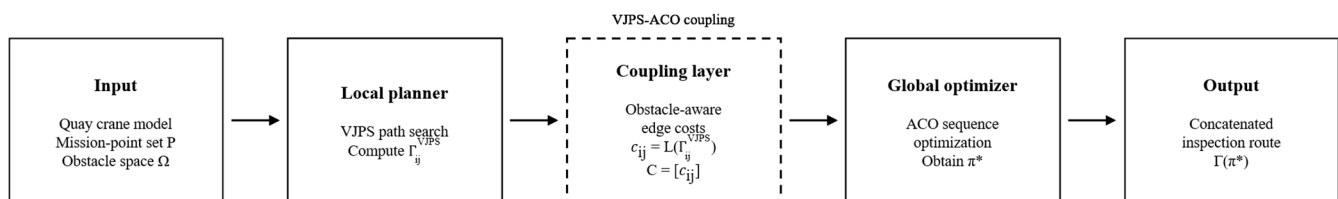


Figure 2. Overall framework of the proposed VJPS-ACO hybrid path planning method. The asterisk denotes the optimized inspection sequence or solution.

4.1. Local Path Planning Using Vector Jump Point Search

This section presents the UAV local path planning method based on the vector jump point search (VJPS) algorithm. The objective is to calculate the shortest collision-free path between each two mission points in the quay crane environment, and the collision-free path length between two points will be used as the initial information for global path planning in Section 4.

4.1.1. Necessity of Local Path Planning

When the UAV flies around a quay crane to perform inspection tasks, it is necessary to avoid collision with the quay crane; thus, a suitable path planning algorithm should be adopted to generate a collision-free path between two points. As shown in Figure 3, the magenta points represent the mission points given by humans, the blue dotted line (path1) represents the path that encounters the obstacle, while the green line (path2) represents a reasonable collision-free path.

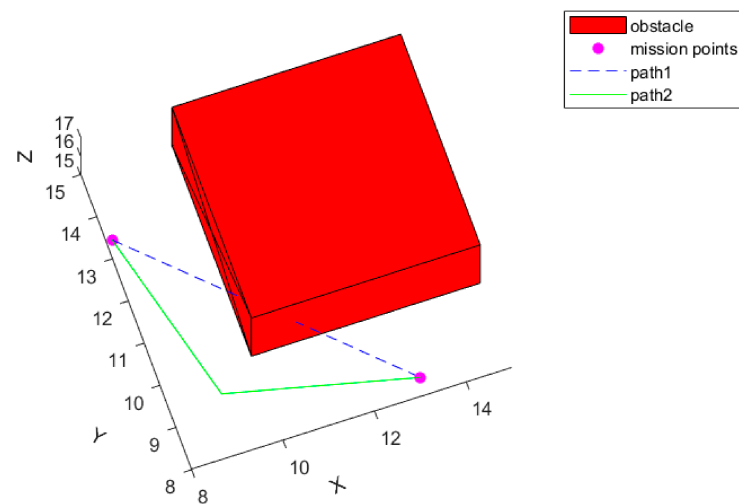


Figure 3. Effect of local path planning.

4.1.2. Jump Point Search

The jump point search (JPS) algorithm is a new heuristic algorithm optimized for the A* algorithm, which is suitable for quickly planning the collision-free path between two points. There are two sets of core rules: the pruning rule and the jumping rule.

Pruning Rule

Given a node x , reached via a parent node p , we prune from the neighbor nodes of x any node n for which one of the following rules applies:

- (1) there exists a path $\pi' = \langle p, y, n \rangle$ or simply $\pi' = \langle p, n \rangle$ that is strictly shorter than the path $\pi = \langle p, x, n \rangle$;
- (2) there exists a path $\pi' = \langle p, y, n \rangle$ with the same length as $\pi = \langle p, x, n \rangle$ but π' has a diagonal move (2-D diagonal move or 3-D diagonal move) earlier than π .

Two types of neighbor nodes remain after applying the pruning rule, and they are called natural neighbor nodes and forced neighbor nodes.

Natural neighbor nodes: The natural neighbor nodes of node x are the remaining neighbor nodes after applying the pruning rule, when the neighbor nodes of x contain no obstacles.

As shown in Figure 4, we draw a $3 \times 3 \times 3$ voxel grid as three 3×3 2-D layers—bottom (B), middle (M), top (T). The current node located in the center of the voxel grid is marked in red, the neighbor nodes of the current node are represented by integer coordinate (x, y, z) , and the natural neighbor nodes of the current node are marked in white. The pruned neighbor nodes are marked in gray. The blue arrow also shows the direction of travel from its parent node, including three cases: (a) straight move; (b) 2-D diagonal move; and (c) 3-D diagonal move.

Forced neighbor nodes: Node n is a forced neighbor node when n satisfies the following requirements:

1. Node n is not a natural neighbor node of x .
2. Node n is a remaining node after the application of the pruning rule.

As shown in Figure 5, when the current node is adjacent to an obstacle (black), the forced neighbor nodes (magenta) cannot be pruned. For example, in case (a) (straight move), voxel $(0, -1, 0)$ and $(0, -1, -1)$ are occupied (black), then voxel $(1, -1, 0)$ and $(1, -1, -1)$ are the corresponding forced neighbor nodes. In case (b) 2-D (diagonal move), the occupied voxel $(0, -1, 0)$ and $(0, -1, -1)$ result in one forced neighbor node $(1, -1, 0)$ and similarly in case (c) (3-D diagonal move).

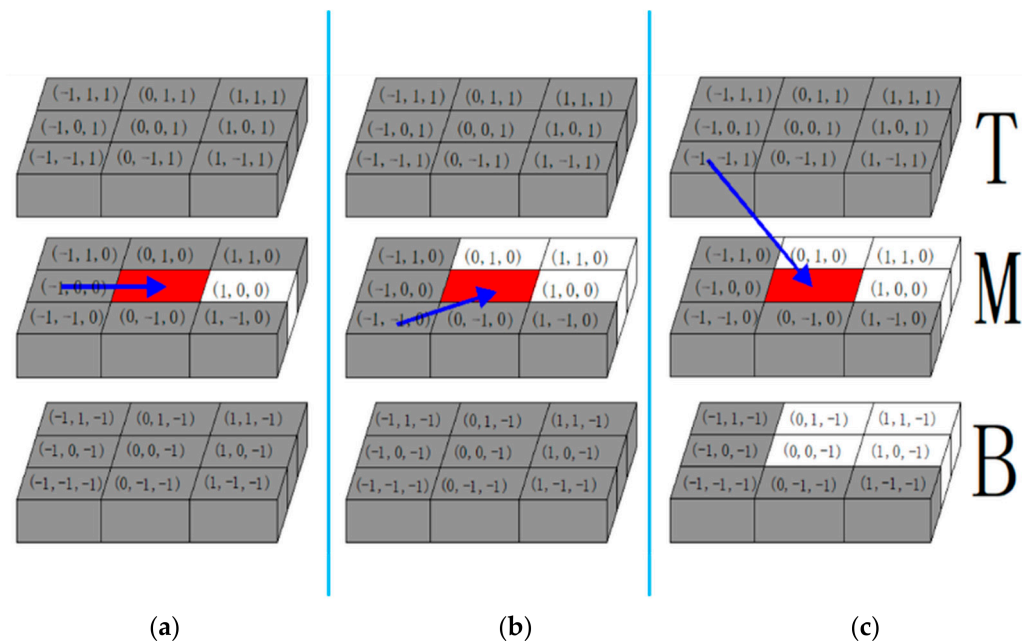


Figure 4. Natural neighbor nodes of each case. (a) Straight move; (b) 2-D diagonal move; (c) 3-D diagonal move. The red voxel denotes the current node, white voxels denote natural neighbor nodes, gray voxels denote pruned neighbor nodes, and blue arrows denote the incoming travel direction.

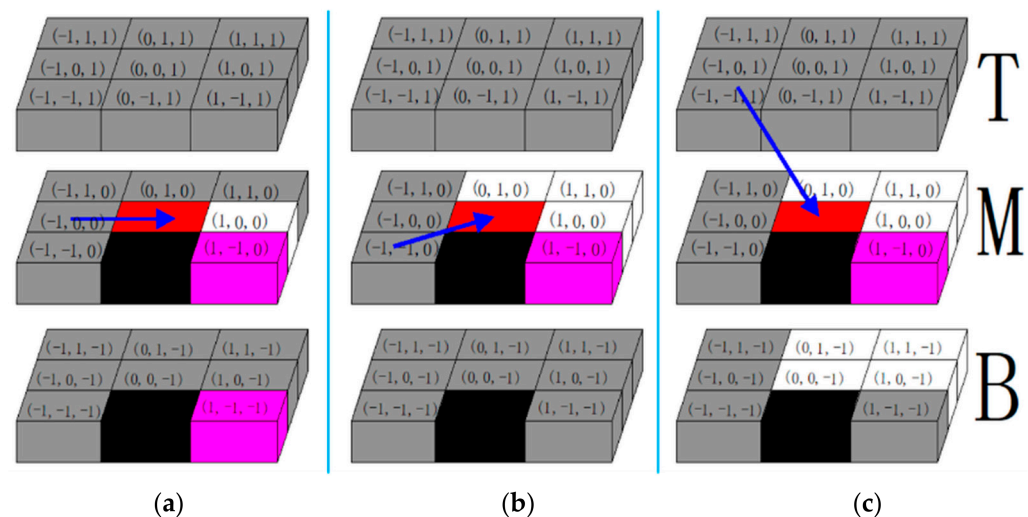


Figure 5. Forced neighbor nodes of each case. (a) Straight move; (b) 2-D diagonal move; (c) 3-D diagonal move. Black voxels denote obstacles, magenta voxels denote forced neighbor nodes, and blue arrows denote the incoming travel direction.

Jumping Rule

In a 3-D environment, JPS applies to each neighbor node of the current node n a simple recursive “jumping” procedure. The objective is to replace each neighbor node of node n with an alternative jump point successor n' that is further away.

According to the relative position between the current node n and the reserved neighbor nodes (natural neighbor nodes and forced neighbor nodes), we can determine all the expansion directions of the current node (one-dimensional straight direction, two-dimensional diagonal direction, three-dimensional diagonal direction). Each expansion direction may result in a jump point successor of the current node. We list the following three cases, which will generate jump point successors of the current node.

(1) If the current node n_1 reaches the goal node g or node n_2 with the forced neighbor nodes according to an expansion direction, then node g or node n_2 will be the jump point successor of the current node n_1 .

(2) When the current node n_1 reaches node n_3 in a two-dimensional diagonal direction, if node n_3 reaches the goal node g or node n_4 with the forced neighbor nodes according to a one-dimensional straight direction, then node n_3 will be the jump point successor of the current node n_1 .

(3) When the current node n_1 reaches node n_5 in a three-dimensional diagonal direction, if node n_5 reaches the goal node g or node n_6 with the forced neighbor nodes according to a two-dimensional diagonal direction or a one-dimensional straight direction, then node n_5 will be the jump point successor of the current node n_1 .

4.1.3. Vector Jump Point Search

In a 3-D environment, using the traditional jump point search (JPS) algorithm will consume unnecessary time to find all the successor nodes of the start node. As shown in Figure 6, there are 26 nodes (white) around the starting node (red). To distinguish the different neighbor nodes, each neighbor node in the grid is represented by its integer coordinate (x, y, z) . For example, the neighbor node labeled $(-1, 1, 1)$ denotes that the neighbor node's coordinate relative to the current node is $(-1, 1, 1)$.

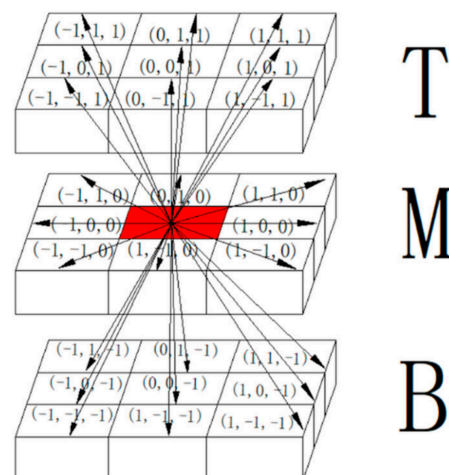


Figure 6. Initial search directions of JPS. The red voxel denotes the starting node, and the black arrows denote the initial search directions.

As shown in Figure 6, JPS explores 26 directions (6 one-dimensional straight directions, 12 two-dimensional diagonal directions, 8 three-dimensional diagonal directions) at the initial stage, but some directions will lead to dead ends. Based on the pruning rule and the jumping rule of JPS, we add two new rules: the preprocessing rule and the simplifying rule. The differences between JPS and VJPS are shown in Figure 7.

In the following section, we will illustrate the proposed preprocessing rule and simplifying rule.

Preprocessing Rule

There are 26 nodes around the starting node at the initial stage, which means that there are 26 directions to expand. We prune some initial search directions according to the location of the goal node (G) relative to the starting node (S), and we label the vector from S to G as $\vec{SG}$. The proposed preprocessing rule's flow chart is shown in Figure 8. In the following, some examples of this preprocessing are described.

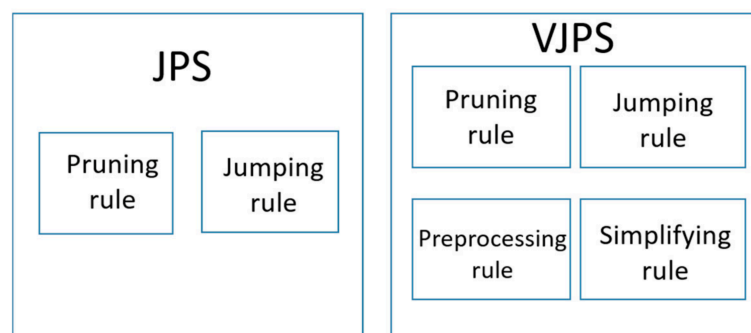


Figure 7. Differences between JPS and VJPS.

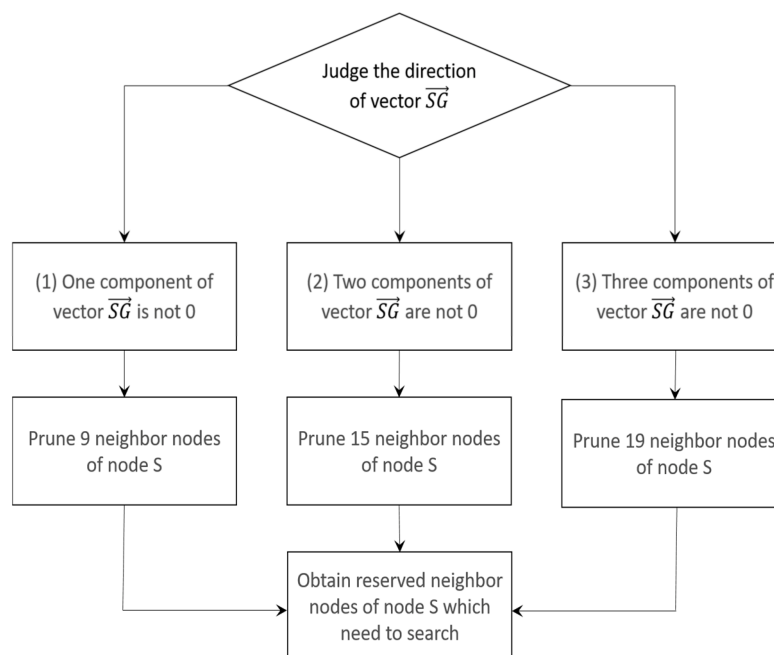


Figure 8. Flow chart of the preprocessing rule.

Case 1: One component of $\vec{SG}$ is not 0

In this case, the algorithm can prune 9 neighbor nodes around the starting node. As shown in Figure 9a, the coordinate of the starting node (S) is (116, 102, 56), the coordinate of the goal node (G) is (154, 102, 56), therefore vector $\vec{SG}$ is (38, 0, 0). The x-component of $\vec{SG}$ being positive, we prune the neighbor nodes of S whose x-component of relative coordinate is negative. As shown in Figure 9b, the neighbor nodes with relative coordinates $(-1, 1, 1)$, $(-1, 0, 1)$, $(-1, -1, 1)$, $(-1, 1, 0)$, $(-1, 0, 0)$, $(-1, -1, 0)$, $(-1, 1, -1)$, $(-1, 0, -1)$, and $(-1, -1, -1)$ can be pruned, and we just need to search 17 directions at the initial stage.

Case 2: Two components of $\vec{SG}$ are not 0

In this case, the algorithm can prune 15 neighbor nodes around the starting node. As shown in Figure 10a, the coordinate of the starting node (S) is (116, 102, 56), the coordinate of the goal node (G) is (154, 118, 56), and the vector $\vec{SG}$ is then (38, 16, 0). The x-component and y-component of $\vec{SG}$ being positive, we prune the neighbor nodes of S whose x-component or y-component of relative coordinate is negative. As shown in Figure 10b, the neighbor nodes with relative coordinates $(0, 1, 1)$, $(0, 0, 1)$, $(1, 0, 1)$, $(1, 1, 1)$, $(0, 1, 0)$, $(1, 1, 0)$, $(1, 0, 0)$, $(0, 1, -1)$, $(0, 0, -1)$, $(1, 0, -1)$, and $(1, 1, -1)$ are reserved after pruning 15 neighbor nodes. We just need to search 11 directions at the initial stage.

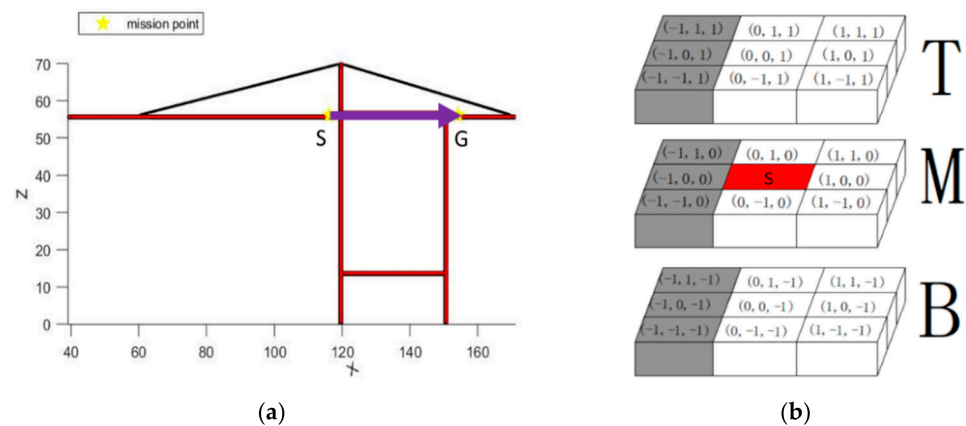


Figure 9. Case 1. (a) The starting node and the goal node; (b) pruned neighbor nodes of the starting node (red). Gray nodes are pruned, white nodes are reserved. The purple arrow denotes the start-goal vector direction.

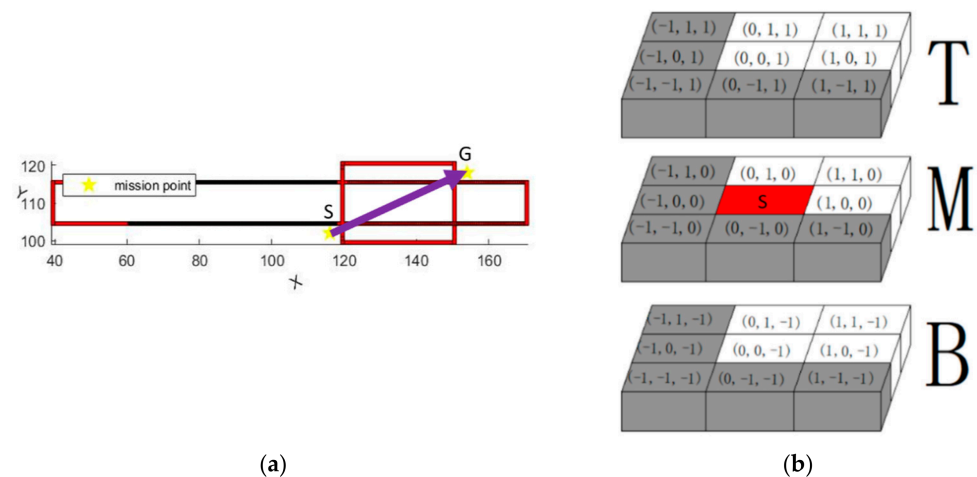


Figure 10. Case 2. (a) The starting node and the goal node; (b) pruned neighbor nodes of the starting node (red). Gray nodes are pruned, white nodes are reserved. The purple arrow denotes the start-goal vector direction.

Case 3: Three components of $\vec{SG}$ are not 0

In this case, the algorithm can prune 19 neighbor nodes around the starting node. As shown in Figure 11a, the coordinate of the starting node (S) is (120, 97, 13), the coordinate of the goal node (G) is (154, 118, 56), and vector $\vec{SG}$ is then (34, 21, 43). All the components of $\vec{SG}$ are positive, we prune the neighbor nodes of S whose component of relative coordinate is negative. As shown in Figure 11b, the neighbor nodes with relative coordinates (0, 1, 1), (0, 0, 1), (1, 0, 1), (1, 1, 1), (0, 1, 0), (1, 1, 0), and (1, 0, 0) are reserved after pruning 19 neighbor nodes. We only need to search 7 directions at the initial stage.

Discussion on optimality and applicability: The preprocessing rule based on the start-goal vector is a heuristic acceleration strategy. Therefore, this study does not claim that VJPS theoretically preserves the complete optimality guarantee of standard JPS in all possible non-convex three-dimensional environments. In highly maze-like environments, an optimal path may require a temporary movement away from the goal direction, and aggressive pruning could theoretically remove such a candidate direction. However, quay cranes are large-scale, sparse, and semi-open steel-frame structures. In this specific application domain, the required detours normally remain consistent with the overall start-goal direction. Thus, VJPS is positioned as a domain-oriented heuristic improvement that

trades general theoretical completeness for higher computational efficiency in structured quay crane inspection scenarios.

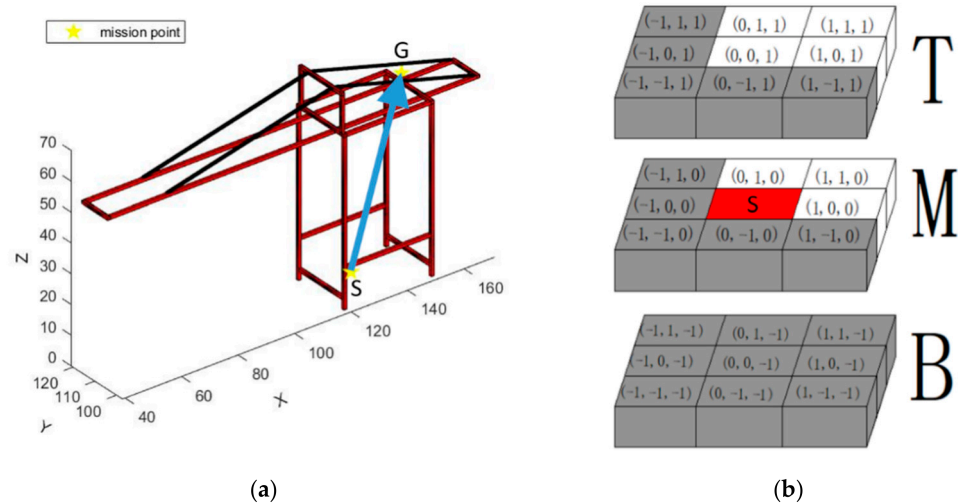


Figure 11. Case 3. (a) The starting node and the goal node; (b) pruned neighbor nodes of the starting node (red). Gray nodes are pruned, white nodes are reserved. The purple arrow denotes the start-goal vector direction.

Simplifying Rule

When the JPS algorithm reaches the goal point, one can get a collision-free path from the starting point to the goal point. As shown in Figure 12, the collision-free path (gray) has 6 waypoints. If the adjacent three waypoints are not in a straight line, then the intermediate waypoint are called the turning point. For example, the adjacent point 2, point 3, and point 4 are not in a straight line, then the intermediate point 3 is called the turning point of the corresponding path. Repeating the action until all the turning points are found, we find out that point 3 and point 4 are the turning points of the original path.

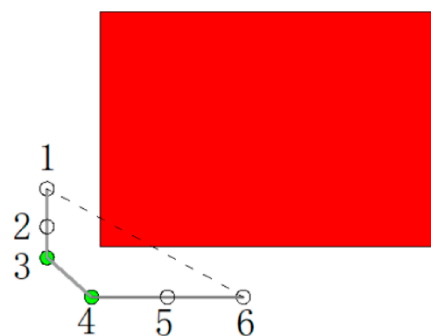


Figure 12. Turning points (green) of the original path (gray). The numbers denote the waypoint order along the path.

To generate the path with fewer turning points, we delete the unnecessary turning points. Let W denote the set of all waypoints, label *keep* as the point to keep, and label Λ as the set of the simplified waypoints; $num(W)$ indicates the number of all waypoints, $collision(a, b)$ is used to judge whether the straight-line path ab intersects with any obstacle. The simplifying function is shown in Algorithm 1.

During path simplification, each candidate straight-line segment is checked in the three-dimensional voxel environment before the intermediate waypoint is removed. A waypoint is deleted only when the direct segment between the retained waypoints does not intersect any occupied voxel. In practical UAV deployment, the occupied voxels can be

inflated according to the UAV size, localization error, and required safety margin so that the simplified path still maintains a safe clearance from the crane structure. Therefore, the simplifying rule is used only after collision detection and should not be interpreted as a purely geometric shortcut without obstacle verification.

Algorithm 1. Simplifying rule pseudocode

```

1. function simplifying(W)
2.    $\Lambda \leftarrow W(1)$ 
3.   keep  $\leftarrow W(1)$ 
4.   for  $i = 2: \text{num}(W) - 1$  do
5.     if collision(keep,  $W(i)$ ) = 0 and collision(keep,  $W(i + 1)$ ) = 1 then
6.       keep  $\leftarrow W(i)$ 
7.        $\Lambda \leftarrow [\Lambda, \text{keep}]$ 
8.     end if
9.   end for
10.   $\Lambda \leftarrow [\Lambda, W(i + 1)]$ 
11.  return  $\Lambda$ 
12. end function

```

As shown in Figure 13, the gray line represents the path generated by the JPS algorithm, and the blue line represents the path optimized by the simplifying rule of the VJPS algorithm. Clearly, the path before applying the simplifying rule has two turning points (point 3 and point 4), while the simplified path has one turning point (point 4).

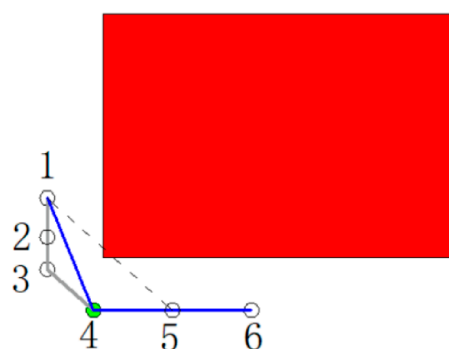


Figure 13. Turning points (green) of the simplified path (blue). The numbers denote the waypoint order along the path.

4.1.4. Steps of the VJPS Algorithm

In this section, we specify the steps of the VJPS algorithm for the local path planning.

Step 1: Construct two data structures: open list and closed list. The open list is used to store the nodes to be expanded, and the closed list is used to store the nodes that have been expanded.

Step 2: Insert the starting node into the open list and clear the closed list.

Step 3: If the open list is empty, then the algorithm will end with failure; otherwise, continue to the next step.

Step 4: Calculate the cost value $f(n)$ of all the nodes in the open list, remove the node with the smallest cost value from the open list and take it as the current node n , then insert it into the closed list. The cost function $f(n)$ is presented as:

$$f(n) = g(n) + h(n), \quad (7)$$

where $f(n)$ is the cost value of the path which starts from the starting node S , visits the current node n , and finally reaches the goal node G ; $g(n)$ is the actual cost of the path from the node S to the node n ; $h(n)$ is the estimated cost of the path from the node n to the node G . This paper uses the Euclidean distance between nodes as $h(n)$, presented as:

$$h(n) = \sqrt{(x_n - x_G)^2 + (y_n - y_G)^2 + (z_n - z_G)^2}, \quad (8)$$

where x_n, y_n, z_n are the coordinate components of the current node n , x_G, y_G, z_G are the coordinate components of the goal node G .

Step 5: If the current node n is the goal node G , go to step 11, else, go to step 6.

Step 6: If the current node n is the starting node S , go to step 7, else, go to step 8.

Step 7: Obtain the neighbor nodes of the current node n according to the pruning rule, then go to step 9.

Step 8: Obtain the neighbor nodes of the current node n according to the preprocessing rule, then go to step 9.

Step 9: Obtain all the jump point successors of the current node n according to the jumping rule.

Step 10: For all the jump point successors of the current node n :

1. If the jump point successor is not in the open list or the closed list, add it to the open list, set the current node n as its parent node, and then go to step 3 and continue the loop.
2. If the jump point successor is already in the open list, calculate the successor's new cost value and compare it with the original cost value. If the new cost value is smaller, record the new cost value as the successor's value, set its parent node as the current node n , and then go to step 3 and continue the loop.
3. If the jump point successor is in the closed list, discard the successor, then go to step 3 and continue the loop.

Step 11: Generate a collision-free path according to the information of the goal node's parent nodes.

Step 12: Simplify the path using the simplifying rule, then output the final path.

The simplified flow chart of the VJPS algorithm is shown in Figure 14. Contents selected by the red rectangles correspond to step 6, step 8, and step 12, which are actually the differences between the JPS algorithm and the proposed VJPS algorithm.

4.2. Global Path Planning Using Ant Colony Optimization

In the actual quay crane inspection task, the UAV not only needs to quickly plan an obstacle-free inspection path between every two mission points, but also needs to plan an appropriate inspection sequence, aiming at generating a short route passing through all the mission points. Planning the inspection sequence of UAV belongs to the problem of global path planning. To solve this problem, the path cost matrix is essential and should be quickly calculated using the VJPS algorithm, and the calculation method is shown in Algorithm 2. The proposed framework adopts a decoupled approach: generating the local obstacle-aware cost matrix first and then optimizing the inspection sequence at the global level. This hierarchical structure is chosen to ensure computational responsiveness and modularity. Although a fully coupled approach might potentially produce a shorter route by dynamically adjusting local paths according to the global sequence, it would substantially increase computational complexity. Therefore, the proposed framework first computes feasible and short collision-free segments between mission-point pairs and then uses these segments as the edge costs for the global sequence optimizer.

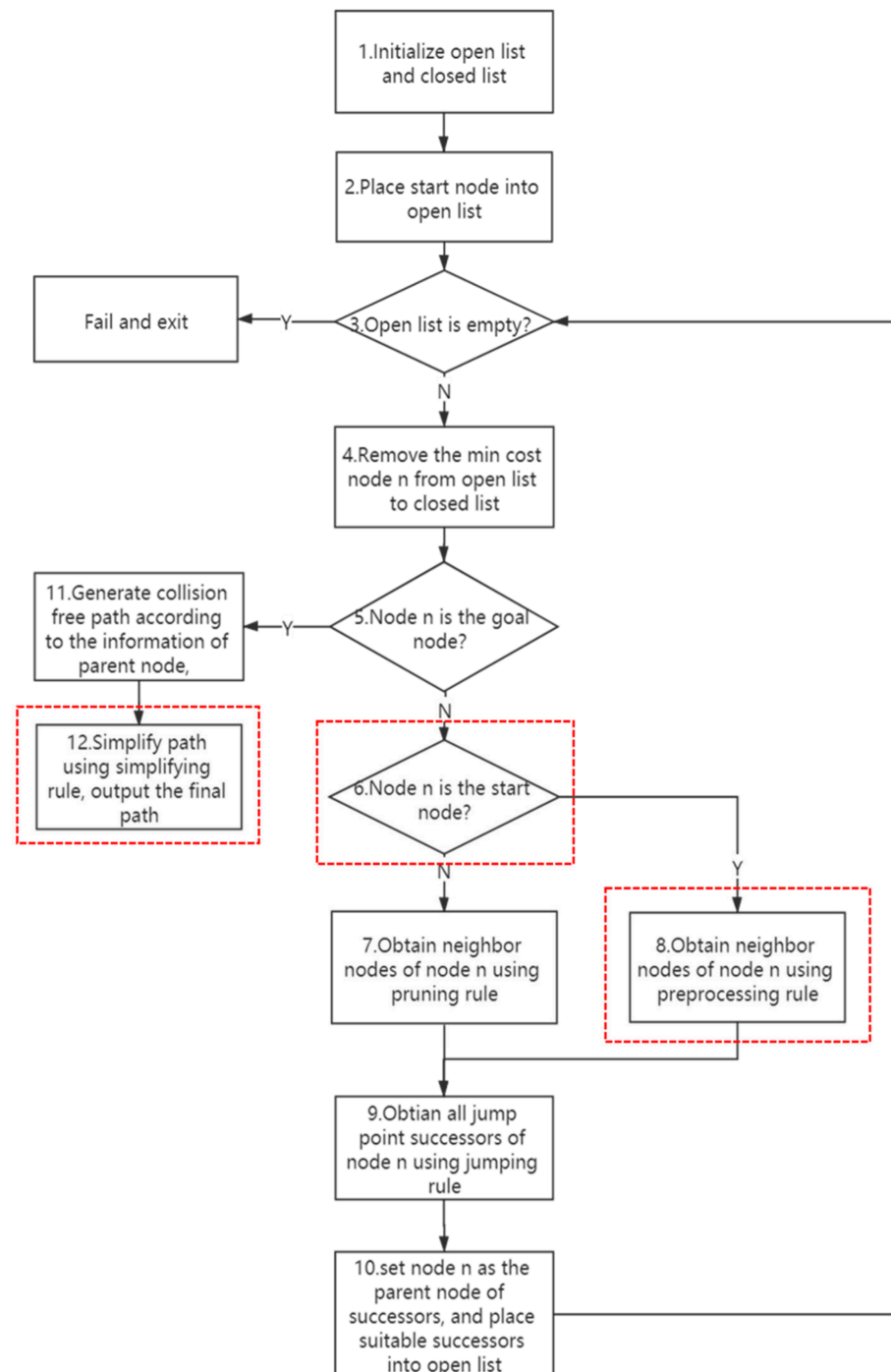


Figure 14. Flow chart of the VJPS algorithm. Contents selected by the red boxes are unique for VJPS compared with JPS.

After calculating the path cost matrix, the ACO algorithm is used to determine an optimized inspection sequence. The simplified flow chart of the ACO algorithm for the UAV global path planning is shown in Figure 15, and the detailed steps of the ACO algorithm are as follows.

Step 1: Obtain the path cost matrix D , which is a 25×25 matrix (25 mission points). The element d_{ij} in D is the length of the collision-free path between mission points i and j , which has been calculated by the VJPS algorithm.

Algorithm 2. Path cost matrix calculation pseudocode

```

1.  $i, j \leftarrow 1$ 
2. while  $i \leq N$  do
3.    $l(i,j) \leftarrow 0$ 
4.    $j \leftarrow i + 1$ 
5.   while  $j \leq N$  do
6.     if there is an obstacle between point  $i$  and point  $j$  then
7.       calculate  $l(i,j) = l(j,i)$  using the VJPS algorithm
8.     else
9.        $l(i,j) = l(j,i) = \sqrt{(\Delta x^2 + \Delta y^2 + \Delta z^2)}$ 
10.     $j \leftarrow j + 1$ 
11.  end while
12.   $i, j \leftarrow i + 1$ 
13. end while
14. return path cost matrix  $D$ 

```

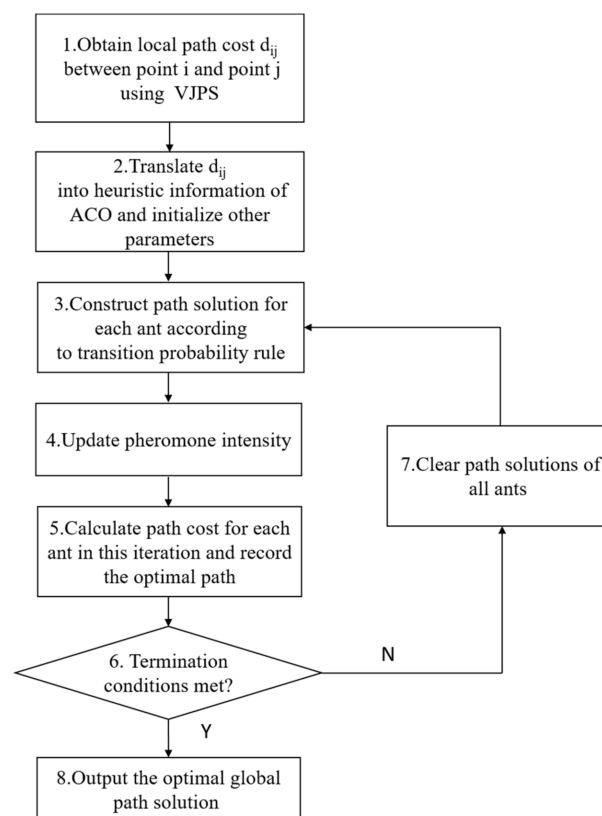


Figure 15. Flow chart of the ACO algorithm for global path planning.

Step 2: Generate the heuristic information η_{ij} between the points i and j according to d_{ij} , then initialize other ACO parameters. η_{ij} is calculated as follows:

$$\eta_{ij} = \frac{1}{d_{ij}}, \quad (9)$$

Step 3: Randomly scatter all ants at 25 points, then put the corresponding starting point into each ant's current solution set. All ants choose their next points based on the

transition probability. At the $t - th$ iteration of the algorithm, the probability for ant k choosing the path from points i to j is calculated as follows:

$$P_{ij}^k(t) = \begin{cases} \frac{\tau_{ij}^\alpha(t)\eta_{ij}^\beta}{\sum_{s \in allowed_k} \tau_{is}^\alpha(t)\eta_{is}^\beta} & j \in allowed_k, \\ 0 & otherwise \end{cases}, \quad (10)$$

where $P_{ij}^k(t)$ is the transition probability of ant k between points i and j at the $t - th$ iteration, $\tau_{ij}(t)$ is the pheromone between points i and j at the $t - th$ iteration, α and β control the relative importance of the pheromone and the heuristic information, η_{ij} is the heuristic information, $allowed_k$ is the selectable points group that ant k can choose.

Step 4: Calculate each ant's path length after visiting all the mission points, and save the shortest path length.

Step 5: Perform the pheromone update. The calculation of pheromone on the path from points i to j can be expressed by:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}(t), \quad (11)$$

$$\Delta\tau_{ij}(t) = \sum_{k=1}^m \Delta\tau_{ij}^{(k)}(t), \quad (12)$$

$$\Delta\tau_{ij}^{(k)}(t) = \frac{Q}{L^{(k)}(t)}, \quad (13)$$

where ρ is the pheromone volatilization coefficient, $\Delta\tau_{ij}^{(k)}(t)$ is the pheromone release of ant k between points i and j at the $t - th$ iteration, m represents the number of ants, Q is the pheromone intensity, and $L^{(k)}(t)$ is the evaluation value of ant k after visiting all the mission points.

Step 6: If stopping criteria are satisfied, go to Step 8, else go to Step 7.

Step 7: Clear the path solution generated by each ant and go back to Step 3.

Step 8: Output the global path of the UAV passing through all the mission points.

In this paper, the ants' population size m is 20, the weight of the ant pheromones is 1, the weight of the heuristic function is 5, the pheromone volatilization coefficient is 0.2, and the pheromone intensity is 10. The algorithm stops when the maximum iteration number is reached or when the best path length remains unchanged for successive iterations.

4.3. Integrated VJPS-ACO Framework and Methodological Contribution

The proposed method integrates the two planning stages through the path cost matrix rather than treating VJPS and ACO as two unrelated procedures. Let P denote the set of inspection mission points and let Ω denote the three-dimensional obstacle space occupied by the quay crane structure. For each ordered pair of mission points, VJPS first generates a local collision-free path in the obstacle-constrained environment.

The coupling mechanism and the information transfer from VJPS to ACO can be expressed as follows:

$$P = \{p_1, p_2, \dots, p_n\}, \quad \Omega \subset \mathbb{R}^3 \quad (14)$$

$$\Gamma_{ij}^{\text{VJPS}} = \text{VJPS}(p_i, p_j, \Omega), \quad i \neq j \quad (15)$$

$$c_{ij} = L(\Gamma_{ij}^{\text{VJPS}}), \quad C = [c_{ij}]_{n \times n} \quad (16)$$

$$\pi^* = \underset{\pi}{\operatorname{argmin}} \left(\sum_{r=1}^{n-1} c_{\pi_r, \pi_{r+1}} + c_{\pi_n, \pi_1} \right) \quad (17)$$

$$\Gamma^* = \bigcup_{r=1}^{n-1} \Gamma_{\pi_r, \pi_{r+1}}^{\text{VJPS}} \cup \Gamma_{\pi_n, \pi_1}^{\text{VJPS}} \quad (18)$$

In Equations (14)–(18), $\Gamma_{ij}^{\text{VJPS}}$ denotes the local path generated by VJPS from p_i to p_j , c_{ij} is the corresponding collision-free path length, C is the obstacle-aware cost matrix, and π^* is the visiting order obtained by ACO. Therefore, the role of VJPS is not only to produce individual local paths but also to transform the three-dimensional collision-constrained environment into a weighted complete graph for global optimization.

Based on C , ACO searches for the inspection sequence that minimizes the accumulated route cost. Once the sequence is obtained, the final inspection route is constructed by concatenating the local VJPS path segments according to this sequence. In this way, VJPS provides feasible obstacle-aware edge weights and ACO performs global sequence optimization over the resulting weighted graph.

This formulation clarifies the hybrid mechanism of the proposed method. Because the local and global stages remain hierarchical, the method should not be interpreted as a formal proof of global trajectory optimality in continuous flight space. For real UAV flight, additional trajectory smoothing, kinematic constraints, safety margins, and real-time obstacle avoidance should be incorporated before deployment.

5. Evaluation and Experimental Testing

In this section, the path planning approach based on VJPS and ACO is implemented in MATLAB R2023b (The MathWorks, Inc., Natick, MA, USA) on a 2.2 GHz PC with a Windows operating system. The simulations are carried out in two parts: local collision-free path planning between mission-point pairs and global inspection-sequence optimization based on the VJPS-generated path cost matrix. Path length is reported in meters and computation time is reported in seconds.

Local Path Planning Experiment

In the local path planning phase, a collision-free path is planned between every ordered pair of mission points. Because 25 mission points are specified for UAV quay crane inspection, 24 directed paths are planned from each mission point to the remaining points, resulting in $25 \times 24 = 600$ directed local paths. The values reported in Table 2 represent the average results over these 600 paths, including the average path-searching time per local path, average path length, average number of expanded nodes, and average number of turning points.

Table 2. Average performance of the two algorithms over 600 directed local paths (time in seconds and length in meters).

Indicator	JPS (J)	VJPS (V)	Relative Difference (R)
Path-searching time (s)	5.3392	4.2907	19.6%
Path length (m)	65.1667	61.8333	5.1%
Number of expanded nodes	560	466	16.8%
Number of turning points	2	1	50.0%

6. Results and Analysis

6.1. Local Path Planning Results and Analysis

As shown in Table 2, the first column represents the path planning evaluation indicators, including path-searching time (s), path length (m), the number of expanded nodes, and the number of turning points. J represents the simulation results of the JPS algorithm, V represents the simulation results of the VJPS algorithm, and R represents the relative difference between the two algorithms. The reported values are averages over 600 directed local paths.

Compared with the JPS algorithm, the VJPS algorithm reduces 19.6% of the searching time and 16.8% of the number of expanded nodes, because the preprocessing rule of the VJPS algorithm prunes unnecessary search directions at the beginning. Moreover, compared with the JPS algorithm, the VJPS algorithm reduces 5.1% of the path length and 50.0% of the number of turning points because the VJPS algorithm adds the simplifying rule, which can further optimize the path.

We use three scenes to further illustrate the local path planning effects of the two algorithms. The differences in the local paths in three cases are shown in Figure 16, Figure 17, and Figure 18 respectively, and the detailed differences in the two algorithms are given in Table 3.

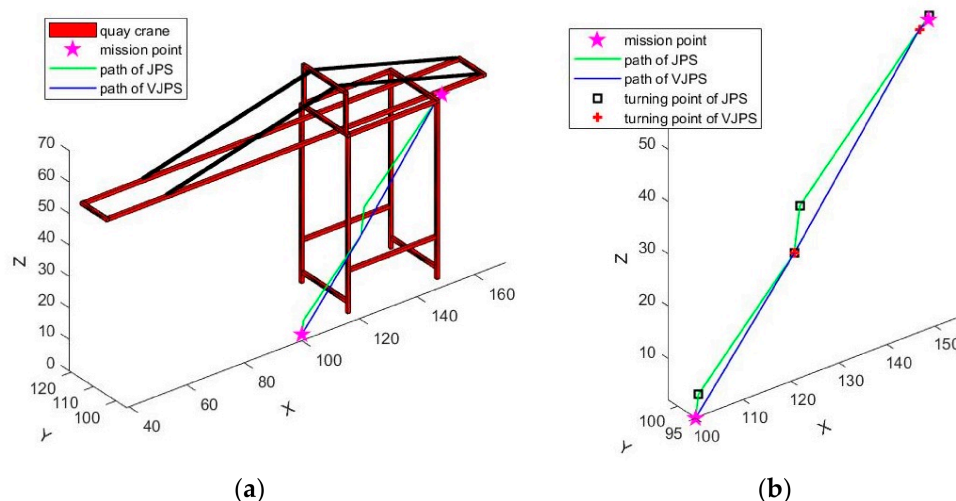


Figure 16. Paths and turning points planned by JPS and VJPS in scene 1 (from mission point 1 to point 5). (a) Different local paths planned by JPS and VJPS; (b) turning points generated by JPS and VJPS.

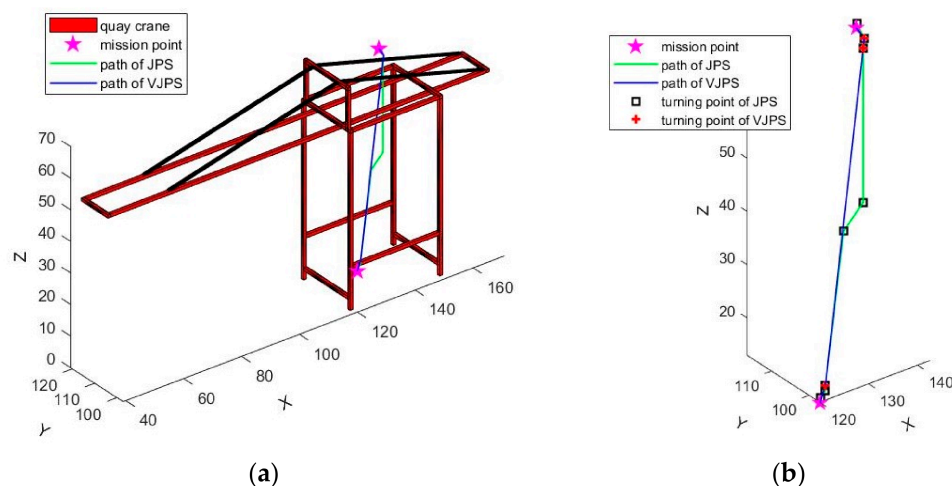


Figure 17. Paths and turning points planned by JPS and VJPS in scene 2 (from mission point 13 to point 18). (a) Different local paths planned by JPS and VJPS; (b) turning points generated by JPS and VJPS.

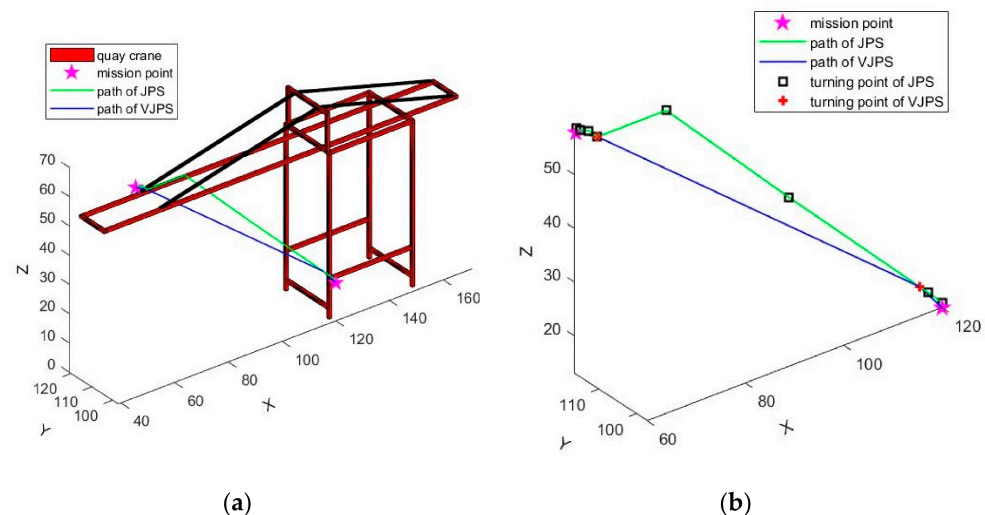


Figure 18. Paths and turning points planned by JPS and VJPS in scene 3 (from mission point 13 to point 21). (a) Different local paths planned by JPS and VJPS; (b) turning points generated by JPS and VJPS.

Table 3. Comparison of the two algorithms in three scenes.

Scene	Simulation Result	JPS (J)	VJPS (V)	Relative Difference (R)
Scene1: from mission point 1 to point 5	Path-searching time (s)	1.8645	0.9464	49.24%
	Path length (m)	80.3244	76.8173	4.37%
	number of expanded nodes	121	119	1.65%
	number of turning points	4	2	50.00%
Scene2: from mission point 13 to point 18	Path-searching time (s)	13.1949	5.8226	55.87%
	Path length (m)	71.0764	62.1597	12.55%
	number of expanded nodes	623	350	43.82%
	number of turning points	8	3	62.50%
Scene3: from mission point 13 to point 21	Path-searching time (s)	19.9112	12.8826	35.30%
	Path length (m)	87.1891	78.3205	10.17%
	number of expanded nodes	792	691	12.75%
	number of turning points	8	2	75.00%

Figure 16 shows the two local paths planned by the two algorithms from mission point 1 to point 5 (scene 1). Figure 17 shows the two local paths planned by the two algorithms from mission point 13 to point 18 (scene 2). Figure 18 shows the two local paths planned by the two algorithms from mission point 13 to 21 (scene 3). Each scene corresponds to two subfigures, where the subfigure (a) represents the different paths planned by the two algorithms and the subfigure (b) shows the turning points planned by the two algorithms. Clearly, compared with the path planned by the JPS algorithm, the path planned by the VJPS algorithm has fewer turning points and a shorter path length.

As shown in Table 3, the path planning effects (i.e., path-searching time, path length, number of expanded nodes, and the number of turning points) of the two algorithms are compared in three scenes. In the three different scenes, the path planned by the VJPS algorithm has fewer turning points and shorter path length compared with the JPS algorithm. In addition, the searching time and the number of expanded nodes of the VJPS algorithm are less than those of the JPS algorithm.

Overall, the VJPS algorithm improves the path quality and searching efficiency in the local path planning phase, establishing a good foundation for the following global path planning.

6.2. Global Path Planning Results and Analysis

After planning a local collision-free path between each two mission points using the VJPS algorithm, the UAV can obtain the information matrix D which includes the actual cost between every two mission points. Table 4 is the path cost matrix, where the first row and the first column denote the number of the mission points (i.e., a total of 25 mission points). We consider the path length planned by the VJPS algorithm as the path cost. For example, the path cost between points 2 and 3 is 32.19, which means that the path length between points 2 and 3 is 32.19 m.

Table 4. Path cost matrix D calculated by the VJPS algorithm.

	1	2	3	4	5	...	25
1	0	17.75	48.41	51.23	76.82		30.25
2	17.75	0	32.19	36.34	65.61		20.00
3	48.41	32.19	0	9.80	53.62		37.51
4	51.23	36.34	9.80	0	45.62		44.32
5	76.82	65.61	53.62	45.62	0		70.67
...							
25	30.25	20.00	37.51	44.32	70.67		0

To determine the final global inspection route of the UAV, a suitable inspection sequence is of great importance. Based on the path cost matrix D calculated by the VJPS algorithm, the inspection path-planning problem can be formulated as a TSP. To evaluate the effectiveness of the proposed ACO algorithm in the global path planning phase, the ACO algorithm is further compared with five methods: the previous method reported in [33], B&B algorithm, PSO algorithm, GA, and SA algorithm. Figure 19 shows the different inspection routes obtained by the different methods. Table 5 lists the corresponding route-comparison results.

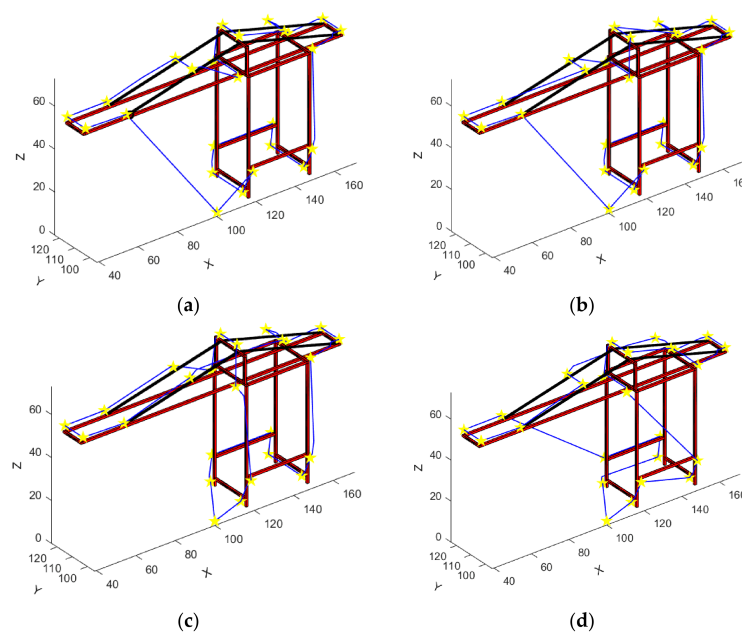


Figure 19. Comparison of global inspection routes obtained by different methods. (a) Inspection route obtained by the ACO algorithm/B&B algorithm; (b) inspection route obtained by the GA; (c) inspection route obtained by the SA algorithm; (d) inspection route obtained by the PSO algorithm. Yellow stars denote mission points, arrows indicate the flight direction, and colors distinguish the plotted route segments.

Table 5. Comparison of global inspection routes obtained by different methods.

	Path Length/m	Iterations	Search Time/s	Obtained the Best Path in This Case?
Previous method [33]	807.3	/	/	No
Inspection route obtained by the PSO algorithm	599.9	399	8.5	No
Inspection route obtained by the SA algorithm	552.3	132	6.4	No
Inspection route obtained by the GA	521.1	120	6.0	No
Inspection route obtained by the ACO algorithm	516.1	72	5.4	Yes
Inspection route obtained by the B&B algorithm	516.1	14,183,010	736.0	Yes

For reproducibility, the comparison algorithms were implemented under the same VJPS-generated path cost matrix and computing environment. The ACO settings were $m = 20$ ants, $\alpha = 1$, $\beta = 5$, $\rho = 0.2$, and $Q = 10$, as described in Section 4.2. For the baseline GA, PSO, and SA algorithms, the standard/default implementation settings were used, and no additional problem-specific parameter tuning was conducted. Their stopping iterations or convergence iterations are reported in Table 5. The B&B method was implemented as a deterministic branch-and-bound search without population-based parameters, and it terminated after completing the corresponding enumeration and pruning process. This clarification is added to make the global comparison easier to understand and reproduce.

As shown in Table 5, compared with the inspection route obtained by the previous method reported in [33], the routes obtained by the other five methods have shorter path lengths. Among these methods, the ACO algorithm and the B&B algorithm obtained the same shortest route length of 516.1 m in this experimental case. However, ACO reached this result with fewer iterations and much less search time than B&B. Because ACO is a heuristic optimization algorithm, this result is described as the best path obtained in this case rather than as a theoretical guarantee of global optimality. This more rigorous interpretation indicates that the proposed VJPS-ACO framework can efficiently generate a high-quality inspection route under the tested quay crane scenario.

6.3. Scalability and Sensitivity Analysis

To verify the robustness of the proposed ACO algorithm, a sensitivity analysis was conducted on the key parameters: the pheromone importance factor and the heuristic importance factor. We tested combinations of the pheromone importance factor in [1,4] and the heuristic importance factor in [1,5]. Simulation results indicated that setting the pheromone importance factor to 1 and the heuristic importance factor to 5 yielded the best convergence speed for the sparse topology of the quay crane inspection graph. Regarding scalability, the present experiments are limited to the 25-point quay crane scenario. Future work will include larger-scale tests with more inspection points and multiple cranes.

7. Discussion

The experimental results demonstrate that the proposed method can effectively improve local path-searching efficiency while supporting global inspection path optimization in the quay crane environment. This improvement can be mainly attributed to the integration of directional path expansion and global heuristic optimization. In the local path-searching stage, the optimized strategy reduces unnecessary node traversal and decreases computational redundancy, thereby improving planning efficiency in the complex structural space of quay cranes. In the global optimization stage, the algorithm further

coordinates multiple inspection points and enhances the continuity of the inspection route, which helps reduce redundant flight distance and unnecessary turning maneuvers.

Compared with conventional heuristic optimization algorithms, such as simulated annealing (SA), genetic algorithm (GA), particle swarm optimization (PSO), and ant colony optimization (ACO), the proposed method shows better adaptability to structured inspection scenarios. Although these traditional algorithms have advantages in global optimization, they may suffer from slow convergence, parameter sensitivity, or limited local obstacle-avoidance capability when applied to dense quay crane structures. By combining improved local path searching with global route optimization, the proposed method provides a more suitable solution for UAV inspection tasks involving multiple key components, narrow spaces, and complex spatial constraints.

From an engineering application perspective, the proposed method offers a feasible path-planning framework for UAV-based quay crane inspection. By optimizing both local movement and the global inspection sequence, the method can potentially reduce inspection time, improve route smoothness, and enhance the operational safety of UAVs around large port machinery. This is particularly meaningful for replacing or supplementing manual inspection in high-risk and difficult-to-access areas, such as booms, trolleys, upper beams, and other elevated crane components.

Nevertheless, several limitations remain. First, the current evaluation is mainly based on simulation experiments, and the constructed environmental model may not fully represent real quay crane operating conditions. In practical port environments, UAV path planning may be affected by wind disturbance, steel-structure occlusion, positioning errors, sensor noise, electromagnetic interference, and dynamic obstacles, including moving spreaders, containers, vehicles, and workers. Second, the present experiments mainly focus on path length and search efficiency, whereas practical inspection tasks also require consideration of image acquisition quality, camera viewing angle, battery endurance, flight stability, and safety margins. Third, the algorithm parameters and inspection-point configurations require further validation under different quay crane types and port layouts.

Therefore, future work should include field experiments on real quay cranes, more comprehensive comparisons with representative path planning algorithms, and robustness tests under complex dynamic operating conditions. In addition, incorporating UAV kinematic constraints, visual inspection requirements, and real-time obstacle avoidance into the proposed framework would further improve its practical applicability in intelligent port inspection.

8. Conclusions

This paper proposed an integrated path planning method for UAV-based quay crane inspection by combining vector jump point search for local collision-free path calculation and ant colony optimization for global inspection-sequence planning. The simulation results show that, compared with the traditional JPS algorithm, the proposed VJPS algorithm reduced the average local path-searching time from 5.3392 s to 4.2907 s, corresponding to a 19.6% improvement, reduced the average local path length by 5.1%, reduced the number of expanded nodes by 16.8%, and reduced the number of turning points by 50.0%. For global inspection planning, the ACO algorithm obtained a 516.1 m inspection route in the tested quay crane scenario, which was shorter than those obtained by SA, GA, PSO, and the previous method.

The study still has several limitations. The evaluation is based mainly on simulation, and the simplified quay crane model does not fully represent wind disturbance, steel-structure occlusion, positioning errors, sensor noise, electromagnetic interference, or dynamic obstacles in real port environments. In addition, the current evaluation focuses

mainly on path length and search efficiency, while practical inspection also requires image acquisition quality, camera viewing angle, battery endurance, flight stability, and safety margins. Future work will include field experiments on real quay cranes, larger-scale tests with more inspection points, comparisons with additional representative path planning algorithms, and the integration of UAV kinematic constraints, visual inspection requirements, and real-time obstacle avoidance.

Author Contributions: Conceptualization, M.W.; resources, M.W.; writing—original draft preparation, M.W.; editing, Z.L. and H.Z. (Haoye Zhang); review and editing, W.Z. and C.C.; visualization, Z.H. and H.Z. (Hongren Zhao); supervision, G.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Acknowledgments: The authors take full responsibility for the content of this publication. All individuals included in this section have consented to the acknowledgement. Microsoft 365 Copilot (latest web-based version; Microsoft 365 Apps Current Channel Version 2605, Build 20026.20168, accessed in June 2026) was used only for grammatical correction, language polishing, manuscript-structure refinement, and improving the clarity of written expression.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Wen, H.; Shi, Y.; Wang, S.; Chen, T.; Di, P.; Yang, L. Route Planning for UAVs Maritime Search and Rescue Considering the Targets Moving Situation. *Ocean Eng.* **2024**, *310*, 118623. [\[CrossRef\]](#)
2. Shao, M.; Wu, B.; Li, Y.; Jiang, X. Research on the Deployment of Professional Rescue Ships for Maritime Traffic Safety under Limited Conditions. *J. Mar. Sci. Eng.* **2024**, *12*, 497. [\[CrossRef\]](#)
3. Liu, J.; Liu, Y.; Ke, Y. Detection and analysis of a quay crane surface based on the images captured by a UAV. *Remote Sens. Lett.* **2020**, *11*, 76–85. [\[CrossRef\]](#)
4. Janoušek, P.; Faigl, J. Speeding Up Coverage Queries in 3D Multi-Goal Path Planning. In Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA), Karlsruhe, Germany, 6–10 May 2013; pp. 5082–5087. [\[CrossRef\]](#)
5. Nikolic, J.; Burri, M.; Rehder, J.; Leutenegger, S.; Huerzeler, C.; Siegwart, R. A UAV System for Inspection of Industrial Facilities. In Proceedings of the 2013 IEEE Aerospace Conference, Big Sky, MT, USA, 2–9 March 2013; pp. 1–8. [\[CrossRef\]](#)
6. Dehghani, M.; Montazeri, Z.; Trojovská, E.; Trojovský, P. Coati Optimization Algorithm: A New Bio-Inspired Metaheuristic Algorithm for Solving Optimization Problems. *Knowl.-Based Syst.* **2023**, *259*, 110011. [\[CrossRef\]](#)
7. Freimuth, H.; König, M. Planning and executing construction inspections with unmanned aerial vehicles. *Autom. Constr.* **2018**, *96*, 540–553. [\[CrossRef\]](#)
8. Englot, B.J.; Hover, F.S. Sampling-Based Coverage Path Planning for Inspection of Complex Structures. In Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS), Atibaia, Brazil, 25–29 June 2012; Volume 22, pp. 29–37. [\[CrossRef\]](#)
9. Tan, Y.; Li, S.; Liu, H.; Chen, P.; Zhou, Z. Automatic Inspection Data Collection of Building Surface Based on BIM and UAV. *Autom. Constr.* **2021**, *131*, 103881. [\[CrossRef\]](#)
10. Applegate, D.L.; Bixby, R.E.; Chvátal, V.; Cook, W.J. *The Traveling Salesman Problem: A Computational Study*; Princeton University Press: Princeton, NJ, USA, 2006. [\[CrossRef\]](#)
11. Cook, W.J. In *Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation*; Princeton University Press: Princeton, NJ, USA, 2012. [\[CrossRef\]](#)
12. Fischetti, M.; Toth, P. An Additive Bounding Procedure for the Asymmetric Travelling Salesman Problem. *Math. Program.* **1992**, *53*, 173–197. [\[CrossRef\]](#)
13. Meer, K. Simulated annealing versus Metropolis for a TSP instance. *Inf. Process. Lett.* **2007**, *104*, 216–219. [\[CrossRef\]](#)
14. Hussain, A.; Muhammad, Y.S.; Sajid, M.N.; Hussain, I.; Shoukry, A.M.; Gani, S. Genetic Algorithm for Traveling Salesman Problem with Modified Cycle Crossover Operator. *Comput. Intell. Neurosci.* **2017**, *2017*, 7430125. [\[CrossRef\]](#) [\[PubMed\]](#)
15. Zhong, Y.; Lin, J.; Wang, L.; Zhang, H. Discrete Comprehensive Learning Particle Swarm Optimization Algorithm with Metropolis Acceptance Criterion for Traveling Salesman Problem. *Swarm Evol. Comput.* **2018**, *42*, 77–88. [\[CrossRef\]](#)

16. Leksakul, K.; Smutkupt, U.; Jintawiwat, R.; Phongmoo, S. Heuristic approach for solving employee bus routes in a large-scale industrial factory. *Adv. Eng. Inform.* **2017**, *32*, 176–187. [\[CrossRef\]](#)
17. Deng, W.; Xu, J.; Zhao, H. An Improved Ant Colony Optimization Algorithm Based on Hybrid Strategies for Scheduling Problem. *IEEE Access* **2019**, *7*, 20281–20292. [\[CrossRef\]](#)
18. Qtaish, A.; Braik, M.; Albashish, D.; Alshammari, M.T.; Alreshidi, A.; Alreshidi, E.J. Enhanced Coati Optimization Algorithm Using Elite Opposition-Based Learning and Adaptive Search Mechanism for Feature Selection. *Int. J. Mach. Learn. Cybern.* **2025**, *16*, 361–394. [\[CrossRef\]](#)
19. Luo, J.; Chen, H.; Xu, Y.; Huang, H.; Zhao, X. An improved grasshopper optimization algorithm with application to financial stress prediction. *Appl. Math. Model.* **2018**, *64*, 654–668. [\[CrossRef\]](#)
20. Sánchez, A.; Cuautle, R.; Zapata, R.; Osorio, M. A Reactive Lazy PRM Approach for Nonholonomic Motion Planning. In *Advances in Artificial Intelligence—IBERAMIA-SBIA 2006*; Bazzan, A.L.C., Labidi, S., Eds.; Springer: Berlin/Heidelberg, Germany, 2006; pp. 542–551. [\[CrossRef\]](#)
21. LaValle, S.M.; Kuffner, J.J., Jr. Randomized kinodynamic planning. *Int. J. Robot. Res.* **2001**, *20*, 378–400. [\[CrossRef\]](#)
22. Nasir, J.; Islam, F.; Malik, U.; Ayaz, Y.; Hasan, O.; Khan, M.; Muhammad, M.S. RRT*-SMART: A Rapid Convergence Implementation of RRT*. *Int. J. Adv. Robot. Syst.* **2013**, *10*, 299. [\[CrossRef\]](#) [\[PubMed\]](#)
23. Dijkstra, E.W. A note on two problems in connexion with graphs. *Numer. Math.* **1959**, *1*, 269–271. [\[CrossRef\]](#)
24. Hart, P.E.; Nilsson, N.J.; Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.* **1968**, *4*, 100–107. [\[CrossRef\]](#)
25. Meng, B.-B.; Gao, X.-G. UAV Path Planning Based on Bidirectional Sparse A* Search Algorithm. In Proceedings of the 2010 International Conference on Intelligent Computation Technology and Automation (ICICTA), Changsha, China, 11–12 May 2010; Volume 3, pp. 1106–1109. [\[CrossRef\]](#)
26. Bolourian, N.; Hammad, A. LiDAR-equipped UAV path planning considering potential locations of defects for bridge inspection. *Autom. Constr.* **2020**, *117*, 103250. [\[CrossRef\]](#)
27. Huang, Z.; Zhai, X.; Wang, H.; Zhou, H.; Zhao, H.; Feng, M. On the 3D Track Planning for Electric Power Inspection Based on the Improved Ant Colony Optimization and A* Algorithm. *Math. Probl. Eng.* **2020**, *2020*, 8295362. [\[CrossRef\]](#)
28. Yao, J.; Lin, C.; Xie, X.; Wang, A.J.; Hung, C.-C. Path Planning for Virtual Human Motion Using Improved A* Star Algorithm. In Proceedings of the 2010 Seventh International Conference on Information Technology: New Generations (ITNG), Las Vegas, NV, USA, 12–14 April 2010; pp. 1154–1158. [\[CrossRef\]](#)
29. Zhan, W.; Wang, W.; Chen, N.; Wang, C. Efficient UAV Path Planning with Multiconstraints in a 3D Large Battlefield Environment. *Math. Probl. Eng.* **2014**, *2014*, 597092. [\[CrossRef\]](#)
30. Harabor, D.; Grastien, A. Online Graph Pruning for Pathfinding on Grid Maps. *Proc. AAAI Conf. Artif. Intell.* **2011**, *25*, 1114–1119. [\[CrossRef\]](#)
31. Liu, S.; Watterson, M.; Mohta, K.; Sun, K.; Bhattacharya, S.; Taylor, C.J.; Kumar, V. Planning Dynamically Feasible Trajectories for Quadrotors Using Safe Flight Corridors in 3-D Complex Environments. *IEEE Robot. Autom. Lett.* **2017**, *2*, 1688–1695. [\[CrossRef\]](#)
32. Almozal, A.; Adil, A.; Feron, E. Safety-Aware Pre-Flight Trajectory Planning for Urban UAVs with Contingency Plans for Mechanical and GPS Failure Scenarios. *Drones* **2025**, *9*, 708. [\[CrossRef\]](#)
33. Tang, G.; Hou, Z.; Claramunt, C.; Hu, X. UAV Trajectory Planning in a Port Environment. *J. Mar. Sci. Eng.* **2020**, *8*, 592. [\[CrossRef\]](#)

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.