Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH Open Access

Grid-Based coverage path planning with NFZ avoidance for UAV using parallel self-adaptive ant colony optimization algorithm in cloud IoT

Yiguang Gong*, Kai Chen, Tianyu Niu and Yunping Liu

Abstract

In recent years, with the development of Unmanned Aerial Vehicle (UAV) and Cloud Internet-of-Things (Cloud IoT) technology, data collection using UAVs has become a new technology hotspot for many Cloud IoT applications. Due to constraints such as the limited power life, weak computing power of UAV and no-fly zones restrictions in the environment, it is necessary to use cloud server with powerful computing power in the Internet of Things to plan the path for UAV. This paper proposes a coverage path planning algorithm called Parallel Self-Adaptive Ant Colony Optimization Algorithm (PSAACO). In the proposed algorithm, we apply grid technique to map the area, adopt inversion and insertion operators to modify paths, use self-adaptive parameter setting to tune the pattern, and employ parallel computing to improve performance. This work also addresses an additional challenge of using the dynamic Floyd algorithm to avoid no-fly zones. The proposal is extensively evaluated. Some experiments show that the performance of the PSAACO algorithm is significantly improved by using parallel computing and self-adaptive parameter configuration. Especially, the algorithm has greater advantages when the areas are large or the no-fly zones are complex. Other experiments, in comparison with other algorithms and existing works, show that the path planned by PSAACO has the least energy consumption and the shortest completion time.

Keywords: Unmanned aerial vehicle, Coverage path planning, Ant colony algorithm, Floyd algorithm, No-fly zone, Grid-based technique

Introduction

With the rapid development of the Internet of Things(IoT), the number of connected devices increases exponentially [1]. Due to the limited energy and computing power, terminal devices often offload computing tasks to servers with higher computing power. One approach is to apply edge computing(EC) technology which can offload tasks to edge servers to relieve the pressure of limited computing resources on terminal devices [2]. Generally, EC arranges edge servers at the network edge so as to ensure that the computation is performed near data sources [3]. The other approach is to offload computing tasks to cloud servers [4]. Cloud computing provides IoT with limitless storage capabilities and computation power [5]. And the blend and incorporation of IoT and Cloud processing forms a new paradigm, named as CloudIoT or Cloud of things (CoT) [6].

The emergence of EC and CloudIoT brings many new applications, such as tasks offloading strategies [7–9], dynamic resource management [10], internet of vehicles [11, 12], geographical Point-of-Interest (POI) recommendation [13], privacy security and recommender systems [14], cloud-based big data technique [15], smart city [16, 17], fault-tolerant placement for cloud systems [18], data forecasting [19], convergence technology of computing, communication and caching [20], edge-cloud collaboration method [21], the efficient data collection with UAVs [22, 23] and et al.. Among them, the application of UAV can significantly improve the capabilities of IoT devices

*Correspondence: yiguang-gong@nuist.edu.cn

School of Automation, Nanjing University of Information Science and Technology, Nanjing, China

by processing the data of these devices. This paper focuses on the application of UAV.

Over the past few decades, UAVs have been widely used in military and civilian applications [24]. The UAVs can be used for search and rescue [25, 26], photogrammetry [27, 28], structures inspection [29, 30], model reconstruction [31, 32], smart farming [33, 34], post-earthquake assessment [35] etc. Many of these UAVs applications involve Coverage Path Planning (CPP) technique, which requires building a path that guarantees that an agent will explore every location in a given scenario [36] . CPP confronts a number of challenges, such as avoiding obstacles [37], data collection [38], avoiding no-fly zone(NFZ) [39], etc. Depending on the size and complexity of areas of interest, exact or approximate cellular decomposition can be applied to decompose the areas and build efficient paths [40]. In addition, planning paths for large areas requires a lot of execution time, using cloud computing is a good choice [41]. For simple CPP missions, the most common performance metrics found in the literature are: the total travelled distance or the path length, the time-to-complete a mission, the area coverage maximization, and the number of turning maneuvers [42]. These metrics depend mainly on two factors: the path length and the number of turns.

This paper, we propose a novel parallel self-adaptive ant colony optimization algorithm(PSAACO) to complete a grid-based CPP. And dynamical Floyd algorithm(DFA) is presented to avoid NFZs efficiently. These two algorithms require a lot of computing time, so they run on the cloud server. Our main contributions are as follows:

- Apply grid-based techniques to decompose the area of interest into cells, map the NFZs using rectangles, and label the gird cells and NFZs with A-Type and N-Type points, respectively. Based on the labeled area, establish the model of the CPP problem, and put forward the formula for the model.
- Improve the ACO algorithm and propose the PSAACO algorithm. Apply the inversion operator and the insertion operator in the PSAACO algorithm, and make it suitable for solving the CPP problem.
- Introduce a self-adaptive setting method for parameters of PSAACO algorithm, and prove the superiority of self-adaptive setting method by experiments.
- Apply multi-thread parallel computing in the algorithm, and prove the improvement of the algorithm performance by experiments.
- Improve the Floyd Warshall algorithm and present the DFA. When vertices are added, only the changes caused by these vertices are calculated in DFA. Apply the DFA to address NFZ avoidance in the CPP problems.
- Compare the PSAACO algorithm with other algorithms and existing works.

The paper is structured as follows: Section 2 describes the related works; Section 3 presents the CPP problem model and provides details about the PSAACO algorithm and DFA; Section 4 presents the performance metrics and experiment results; and Section 5 concludes the paper.

Related Works

CPP is a critical issue for many UAV applications, and it has become also a research hotspot. According to whether the environment is known or not, CPP algorithms can be divided into two categories: online CPP and offline CPP [43]. Offline CPP algorithms only depend on static environmental information, assuming that all environmental information is known in advance. Online CPP algorithms don't need to know the complete information of the environment to be covered in advance, and plan local paths based on real-time sensor information.

According to the employed cellular decomposition technology, CPP algorithms can be divided into three main types: no decomposition, exact cellular decomposition and approximate cellular decomposition [42].

Using a single UAV to perform CPP tasks in a simple area with regular shape usually does not need cell decomposition. Back-and-forth(BF) and Spiral(SP) are two most common no decomposition CPP algorithms [42]. In the BF algorithm, the UAV starts from a certain point on the edge of the area and flies forward along a straight path in a specific direction. After reaching the edge of the area, the UAV flies back in the opposite direction along a parallel path, and the UAV flies back and forth until a path covering the entire area is generated. The specific process of the SP algorithm is as follows: The UAV moves clockwise or counterclockwise along the edge of the unvisited part of the area. If it encounters an obstacle, the UAV rotates clockwise or counterclockwise for a certain angle and then continues to move forward, and so on and so forth until the UAV completes covering the entire area.

Exact cellular decomposition divides the irregularly-shaped complex region into regular-shaped simple cells, and then the path planning is performed on the cells. The classical exact cellular decomposition methods include Trapezoidal Decomposition and Boustrophedon Decomposition [43]. Trapezoid Decomposition is applied on 2D polygonal areas consisting of polygonal boundaries and polygonal obstacles. Starting at each vertex of the boundary and obstacles, draw upper line segment and lower line segment in the non-obstacle part of the area. These line segments decompose the non-obstacle part into simple trapezoidal cells, and then plan paths on these trapezoidal cells. Boustrophedon Decomposition method is similar to the Trapezoid Decomposition, but the two endpoints of the line segment dividing the area need to be located on the boundary of the area. Boustrophedon Decomposition decomposes an area into simple convex polygons, which can be covered by a simple back-and-forth run. Compared with the trapezoidal method, the Boustrophedon Decomposition can reduce the number of cells, thereby shortening the total length of the coverage path.

Approximate cellular decomposition, also known as grid-based decomposition, decomposes an area of interest into regularly-shaped grids. These grids are usually square, but they can be triangular or hexagonal. Each grid is marked with a value to indicate whether there are obstacles in it. Grid-based decomposition tries to find out the optimal path to traverse these grids and avoid obstacles. Grid-based decomposition methods are easy to create, simple and intuitive to use, and are widely used methods at present.

Authors in [44] propose an cost-efficient multirobot CPP algorithm. This method consists of two parts. The first part divides the area of interest into sub-areas. The second part applies a single UAV on each sub-area, and uses an algorithm based on the wave-front planner to plan path for the sub-area. The algorithm creates an adjacency graph of the grids, and then applies a breadth-first search (BFS) on the graph to perform the distance transformation. The path planned by the algorithm takes the starting cell as the basic cell, selects the cell with the smallest gradient rise in the neighborhood of the basic cell as the next point of the path. Take this new path point as the basic cell, and repeat the above process to find the subsequent points for the path until the path covers the entire area.

Authors in [45] propose a method to plan coverage paths on irregular-shaped areas for image mosaicing. The algorithm uses a cost function designed to minimize the number of turns. This method consists of two parts. In the first part, the area is decomposed into regular-shaped grids, then this decomposed area is converted to a regular graph. The second part adopts a method similar to that in literature [44], applies the BFS on the grids to perform the distance transformation, then sequentially selects the cells with the smallest gradient rise in the neighborhood to form a coverage path. Deep-limited search is applied to ensure that the method can obtain a path that traverses all nodes and each node is traversed only once. A backtracking procedure is also used to solve the problems of same potential weight neighbor selection and trapping in local optima.

Cabreira et al. [39] introduce an energy-aware grid-based method. The approach is designed to minimize energy consumption of mapping tasks on the areas with irregular shapes. The method improves the grid-based approach proposed by [45], and proposes a new cost function to replace the original one in [45], which aims to minimize the number of turns. The new energy cost function takes into account not only the energy required for the UAV to turn, but also the energy consumed by the UAV when accelerating, decelerating and flying at a constant speed. The new approach is able to lower energy consumption in real flight experiments.

Ghaddar et al. [46] introduce an obstacle avoidance approach based on energy-aware grids. The method can be divided into two stages. The first stage is to plan one coverage path offline on the area of interest based on the top view girds. The second stage is an online obstacle avoidance process based on the information captured by the camera. If an obstacle is captured , the proposed method re-plans the path to avoid the obstacle. This literature presents the experimental results on two scenarios, showing that the proposed method can shorten the flight time and reduce the energy consumption.

In another paper [40], Ghaddar et al. propose an energy-aware CPP algorithm using grid-based technique to cover areas where NFZs exist. The algorithm includes three main stages. The first stage is to divide the area of interest into grids and graphically represent the NFZs. The second stage is to partition the area into sub areas around the NFZs. And the third stage is to plan one path on each sub area. To increase the percentage of covered area, the proposed algorithm selects the partition borders on the boundaries of cells. This approach applies filtering method to get rid of edges and nodes that are negligible for the coverage task. And this method uses a new cost function to build better turning points selection mechanism. The proposed approach is suitable for scenarios using single and multiple UAVs.

Materials and Methods

In this paper, our goal is to plan a coverage path over an area of interest where NFZs exist. The requirements to fulfill for the CPP are:

- Complete coverage of the area of interest;
- Ensure minimum completion time, by lowering the turning angle and the length of the planned path.
- Avoid passing over the interior of NFZs, but the boundary lines of NFZs are passable;

We use one UAV in offline mode to cover an area where NFZs exist. We build the CPP problem model using grid-based technique. We propose the PSAACO to make path planning on this problem model. Furthermore, to avoid the NFZs, we propose a DFA, which can dynamically add points and only need to calculate the changes caused by these points without recalculating the entire algorithm.

Problem Model

We consider planning a path for an area of size approximately 100m × 80m, the area is shown in Fig. 1a. In the figure, the grey polygon represents the area that needs to be covered and searched, while the red polygons represent the NFZs.

Similarly to [40], this work uses the grid-based technique to model the investigated area. We map the geographical area to grid cells, and the mapped image is shown in Fig. 1b. The longest side of the area is aligned parallel to the X-axis or Y-axis of the mapped image by means of coordinate transformation. Each grid cell has rectangular shape, and its size is determined by the UAV footprint, what it means is the area the UAV can cover at one time [47]. In Fig. 1b, each grid cell is 20 meters long and 20 meters wide. Grid cells filled entirely or partially with grey represent areas that need to be covered, and grid cells filled with red represent NFZs, while the

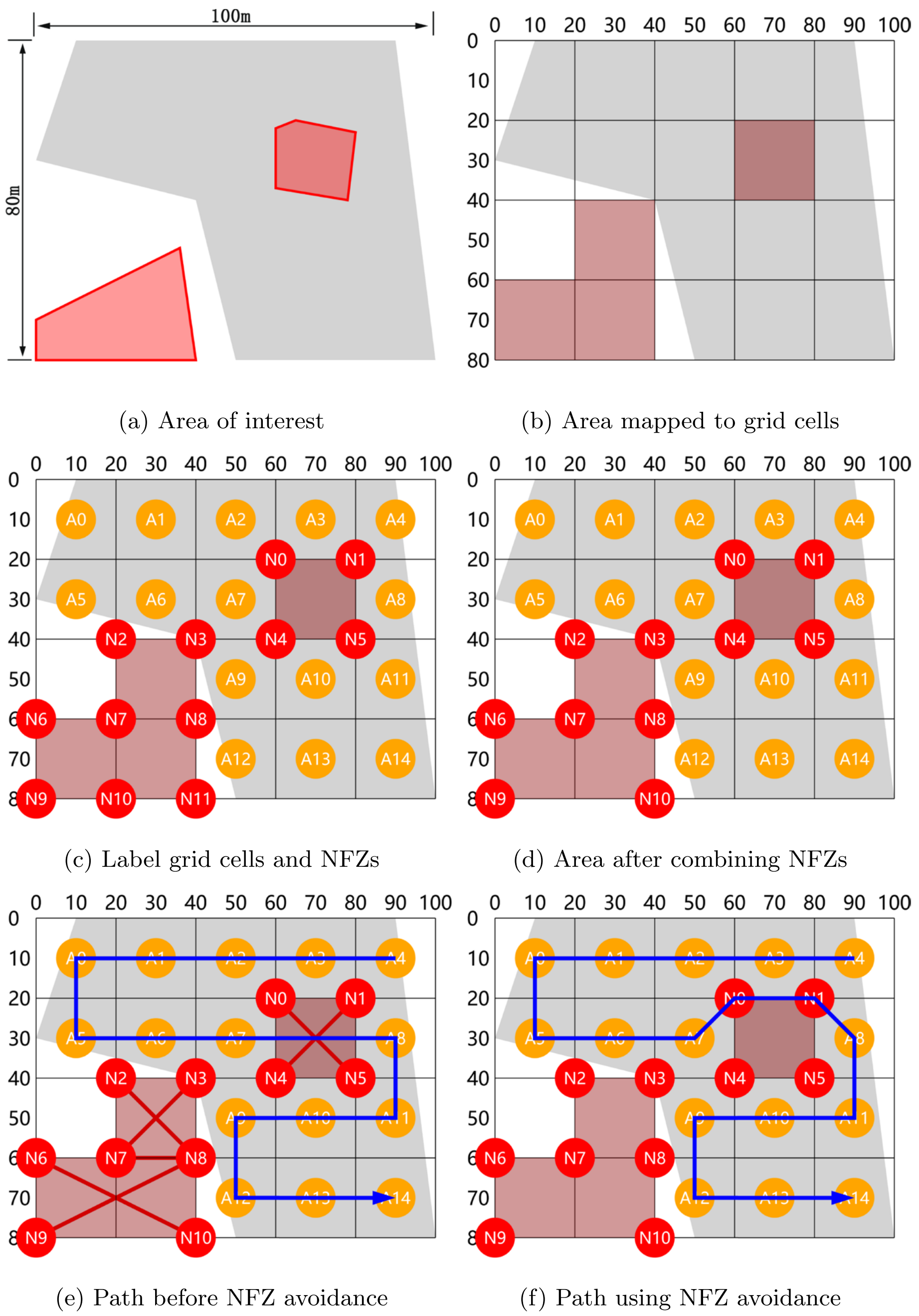

100m
80m
0 10 20 30 40 50 60 70 80 90 100
0 10 20 30 40 50 60 70 80
(a) Area of interest
(b) Area mapped to grid cells
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14
N0 N1 N2 N3 N4 N5 N6 N7 N8 N9 N10 N11
(c) Label grid cells and NFZs
(d) Area after combining NFZs
(e) Path before NFZ avoidance
(f) Path using NFZ avoidance

Fig. 1 Grid-based area decomposition and cells labeling

unfilled cells mean areas that do not need to be covered or are not prohibited to fly over.

In Fig. 1c, we label the key points of the mapped image. The key points are divided into two categories, named as A-Type points and N-Type points, respectively. A-Type points are the center points of the grid cells that need to be covered and searched, marked with orange circular labels in the figure. N-Type points are the vertices of NFZ polygons, and marked with red circular labels. The key points in each row are numbered from left to right (x increasing direction), and the rows are numbered incrementally from top to bottom (y increasing direction). The text in each label consists of its type name and sequentially number.

In this work, a key point is represented as a vector $V(te, sn, x, y)$, where $coord(x, y)$ is the coordinate, te is the type of the point, sn is the sequentially number. For example, in Fig. 1c, point V_{A10} can be represented by a vector $V(“A'', 10, 70, 50)$, while point V_{N7} can be represented by a vector $V(“N'', 7, 20, 60)$.

NFZs are described by rectangles, eg rectangle $NR(V_{N0}, V_{N1}, V_{N5}, V_{N4})$ is an instance of NFZ. If some NFZ rectangles have common edges, try to combine them into a larger rectangle, eg rectangle $NR(V_{N6}, V_{N7}, V_{N10}, V_{N9})$ and rectangle $NR(V_{N7}, V_{N8}, V_{N11}, V_{N10})$ can be combined into rectangle $NR(V_{N6}, V_{N8}, V_{N11}, V_{N9})$. After the NFZs are combined, we can remove those redundant N-Type vertices that are not used to form the NFZ rectangles. Figure 1d shows the area image after removing the redundant vertices.

Figure 1e shows an example of CPP, the back-and-forth is applied on the area. The blue polyline represents the planned path, which starts at point V_{A4} and ends at V_{A14}. In this paper, one path is described by one sequence of points it passes through, and the planned path without considering NFZ avoidance is as follows.

$$P_i = P(V_{A4}, V_{A3}, V_{A2}, V_{A1}, V_{A0}, V_{A5}, V_{A6}, V_{A7}, V_{A8}, V_{A11}, V_{A10}, V_{A9}, V_{A12}, V_{A13}, V_{A14}) \tag{1}$$

In this work, we detect whether all line segments in the path intersect the diagonal of the rectangle. If there is a line segment that intersects the diagonal of the NFZ rectangle, it is considered that the line segment has passed the NFZ, and NFZ avoidance processing is required. Use a set named NFZS to record the diagonals of all NFZ rectangles. In Fig. 1e, the diagonals in NFZS are represented by red line segments, and NFZS is as follows.

$$NFZS = \{LS(V_{N0}, V_{N5}), LS(V_{N1}, V_{N4}), LS(V_{N2}, V_{N8}), LS(V_{N3}, V_{N7}), LS(V_{N6}, V_{N10}), LS(V_{N8}, V_{N9}))\} \tag{2}$$

As a special case, the line segment $LS(V_{N7}, V_{N8})$ is the common edge of two NFZs, which is considered to be the interior of the area. If there is an intersection between the path P_i and $LS(V_{N7}, V_{N8})$, NFZ avoidance is also required. The common edge $LS(V_{N7}, V_{N8})$ is also recorded in NFZS, and the complete NFZS of the Fig. 1e is as follows.

$$NFZS = \{LS(V_{N0}, V_{N5}), LS(V_{N1}, V_{N4}), LS(V_{N2}, V_{N8}), LS(V_{N3}, V_{N7}), LS(V_{N6}, V_{N10}), LS(V_{N8}, V_{N9})), LS(V_{N7}, V_{N8})\} \tag{3}$$

As can be seen from Fig. 1e, the line segment $LS(V_{A7}, V_{A8})$ of the path intersects $LS(V_{N0}, V_{N5})$ in the NFZS , which means that the planned path P_i passes through the NFZ, but this is not allowed.

In this paper, we use N-Type points to avoid NFZs. Two N-Type points, V_{N0} and V_{N1}, are inserted into the line segment $LS(V_{A7}, V_{A8})$. The update path P_j successfully avoids the NFZ, the vector of P_j is as follows, and it is illustrated in Fig. 1f.

$$P_j = P(V_{A4}, V_{A3}, V_{A2}, V_{A1}, V_{A0}, V_{A5}, V_{A6}, V_{A7}, V_{N0}, V_{N1}, V_{A8}, V_{A11}, V_{A10}, V_{A9}, V_{A12}, V_{A13}, V_{A14}) \tag{4}$$

In short, the CPP problem in this paper can be regarded as an optimization problem. The problem is described as follows: plan a path passing through all A-Type points, require each A-Type point to be traversed only once, use N-Type points to avoid the NFZs, and require the shortest time for a UAV to cover the path. The mathematical model of the CPP problem can be described by Eq. 5.

$$\tau = \underset{\forall P_i \in G}{minimize} f(P_i, \nu, \psi) \tag{5}$$

Where f is a function to calculate the completion time for a planned path. τ is the minimum completion time of all paths. P_i is a planning path, as in Eq. 4. ν is UAV speed, and ψ is UAV rotation rate. G means area after decomposition, which is composed of several sets. VAS is the set of A-Type points, containing all A-Type points in the area, while VNS is the set of N-Type points. $NFZS$ defines a set of line segments to determine whether NFZ avoidance is required, which consists of the diagonals of the NFZ rectangles and the common edge of two NFZ rectangles.

$$G = G(VAS, VNS, NFZS) \tag{6}$$

The mathematical model of the CPP problem subject to:

$$\forall V_k \in P_i \rightarrow V_k \in (VAS \cup VNS) \tag{7}$$

$$\forall V_k \in VAS \rightarrow V_k \in P_i \tag{8}$$

$$i \neq j, V_i \in P_k, V_j \in P_k \rightarrow V_i \neq V_j \tag{9}$$

$$\forall LS_x \in P_i, \forall LS_y \in NFZS \rightarrow LS_x \cap LS_y = \varnothing \tag{10}$$

Constraint formula 7 represents any point in the planned path P_i, which must also exist in the set of A-Type points or N-Type points. Constraint formula 8 indicates that P_i contains all A-Type points. Constraint formula 9 shows that each point in P_i can only appear once, and no two identical points exist in the same path. Constraint formula 10 states that the line segment LS_x in P_i does not intersect with the line segment LS_y in NFZS, that is, the path P_i does not pass through the NFZs.

Table 1 defines the notation of relevant sets, vectors, parameters and variables of our work.

PSAACO Algorithm for CPP

Ant colony algorithm(ACO) is a bionic optimization algorithm, which simulates the behavior of ant colony foraging in nature, and uses probabilistic search technology to find the optimal path. This technique is first introduced by Dorigo and his colleagues [48], and then many scholars have continuously enriched and improved it, forming an algorithm family. The well-known algorithms in the algorithm family include: Elitist Ant System (EAS) [49], Rank Based Ant System(ASRank) [50], MAX-MIN Ant System (MMAS) [51], Ant Colony System(ACS) [52], etc. ACO is used for many applications especially complex combinatorial optimization problems such as vehicle routing problem (VRP) [53], job-shop scheduling problem (JSP) [54], traveling salesman problem (TSP) [55], quadratic assignment problem (QAP) [56], and so on.

In this paper, we extend the ACO algorithm presented in our previous work [57], and propose a new PSAACO to deal with CPP problem. This new PSAACO algorithm is mainly expanded as follows: make some modifications to solve CPP problems, modify the self-adaptive strategy to construct and apply a new parameter vector at each iteration, apply the inversion and insertion operators to construct new solutions, use a multi-threaded parallel computing approach to improve the solution quality and search speed.

The pseudocode of the PSAACO is presented in Algorithm 1. As seen in the algorithm, procedure InitializeGlobalParameters() is used to initialize the global parameters G, ψ, ν, $TdNum$, $Itmax$,$Itmax2$, $Cmax$, $Cmin$, a, n and q, whose meanings are provided in the algorithm. The G is an instance of CPP model, as represented by Eq. 6. The CPP model instance contains three sets: A-Type points set VAS, N-Type points set VNS and line segments set $NFZS$. The following subsections elaborate on the other procedures in the algorithm.

Table 1 Table of notations

Symbol	Description	Symbol	Description
A	Geographical area	te	Type of the point
G	Instance of CPP model	sn	Sequentially number of point
V_{Ai}	A-Type point	V_{Ni}	N-Type point
NR	NFZ rectangle	P_i	Planned path
$coord(x, y)$	Coordinate of point	SP_i	Sub-path consisting of partial points of the path
LS_i	line segment	VAS	Set of A-Type points
VNS	Set of N-Type points	$NFZS$	Set of line segments to define NFZs
ν	UAV speed	ψ	UAV rotation rate
β	Total distance of a path	φ	Total turning angles of a path
Ξ	Total energy consumption of a path	τ	Total completion time of a path
S	The solution archive	n	Number of solutions in solution Archive
m	Number of variables in solution vector	s_i	The i-th solution int solution archive
s_i^j	Value of the j-th variable in the i-th solution	ω_i	selection weight of the i-th vector in the archive
η	The size of search solution archiving area	C	The self-adaptive parameter archive
$Cmax$	Array of parameter maximum values	$Cmin$	Array of parameter minimum values
a	Number of parameters in parameter Archive	b	Number of parameters in parameter vector
c_i	The i-th parameter vector int archive	c_i^j	the j-th parameter in the i-th parameter vector
μ	the mean value of Gaussian function	σ	the mean square error of Gaussian function
d_{ij}^l	distance from vertex i to j in the l-th iteration	ξ	The convergence speed of parameter archive
θ	The size of search parameter archiving area	λ	Energy consumption per meter of length
γ	Energy consumption per degree of turn angle	q	Number of ants for each thread
$TdNum$	Number of parallel threads	$Itmax$	Maximum number of iterations
$Itmax2$	Maximum iteration number of finding no better solution		
PM	Sub-paths matrix between any two points in VAS after avoiding NFZs		

Algorithm 1 The pseudocode of PSAACO

```
    Input : The CPP instance and parameter values;
      $G$: Instance of CPP model, $\psi$: UAV rotation rate, $\nu$: UAV speed,
      $TdNum$: Number of parallel threads, $Itmax$: Maximum number of iterations,
      $Itmax2$: Maximum iteration number of finding no better solution,
      $Cmax$: Array of self-adaptive parameter maximum values,
      $Cmin$: Array of self-adaptive parameter minimum values,
      $a$: Number of parameters in parameter Archive,
      $n$: Number of solutions in solution Archive,
      $q$: Number of ants for each thread
    Output : The best path and its complete time;
      $P_{best}$: The best path, $\tau$: The complete time of the best path
 1: begin
    // Initialize global parameters: $G$, $\psi$, $\nu$, et al.
 2: InitializeGlobalParameters()
    // Load the CPP model instance $G$
    // $PM$ is sub-paths matrix between any two points in $VAS$ after avoiding NFZs
 3: $PM$ = CalculateSubpathMatrix($G$)
 4: $bestSoluFit = 0$
 5: $bestSolu$ = null
    // Multiple threads started in parallel
 6: for $tdjs = 1$ to $TdNum$ do
      // Initialize the self-adaptive parameter archive
 7:     $C$ = InitializelParameterArchive ($Cmax$, $Cmin$, $a$)
      // Select a parameter array from the archive as the current parameter array
 8:     $currentPaArray$ = SelectParameter ($C$)
 9:     $n = currentPaArray[1]$
      // The initialization of the solution archive
10:     $S$ = InitializeSolutionArchive ($G$, $n$, $PM$, $\psi$, $\nu$)
      // Sort solutions in the archive by fitness value and remove solutions ranked
    out of $n$
11:     UpdateSolutionArchive($S$, $n$)
12:     $endIt = Itmax2$
13:     $LastPaFit = 0$
      // start iterations
14:     for $itjs = 1$ to $Itmax$ do
        // Select a parameter array from the archive as base parameter array
15:       $basePaArray$ = SelectParameter ($C$)
        // Construct a new parameter array based on the base parameter array
16:       $currentPaArray$ = ConstructParameter ($basePaArray$)
17:       $n = currentPaArray[1]$
18:       $q = currentPaArray[2]$
        // Initialize the fitness value of current parameter array
19:       $PaFit = 0$
        // start ants to construct new solution
20:       for $antjs = 1$ to $q$ do
          // Select a solution from the solution archive as base solution
21:         $baseSolution$ = SelectSolution ($S$, $currentPaArray$)
          // Construct a new solution based on this base solution
22:         ($newSolution$, $newSolutionFit$) = ConstructSolution ($baseSolution$,
    $PM$, $\psi$, $\nu$)
          // Add the new solution into the solution archive
23:         AddSolutionIntoArchive ($newSolution$, $newSolutionFit$)
24:         if $newSolutionFit < PaFit$ or $PaFit == 0$ then
25:           $PaFit = newSolutionFit$
26:         end if
27:       end for
28:       UpdateSolutionArchive($S$, $n$)
        // Add the new parameter into the parameter archive
29:       AddParameterIntoArchive ($currentPaArray$, $PaFit$)
30:       UpdateParameterArchive($C$, $a$)
        // Algorithm termination judgment and parameter update
31:       if $PaFit < LastPaFit$ or $LastPaFit == 0$ then
32:         $endIt = Itmax2$
33:         $LastPaFit = PaFit$
34:       else
35:         $endIt = endIt$-1
36:         if $endIt \leq 0$ then
37:           break
38:         end if
39:       end if
40:     end for
      // Take the solution with the minimum fitness value from the solution archive
41:     ($minSolution$, $minSolutionFit$) = $S[1]$
42:     if $minSolutionFit < bestSoluFit$ or $bestSoluFit == 0$ then
43:       $bestSoluFit = minSolutionFit$
44:       $bestSolu = minSolution$
45:     end if
46: end for
    // Convert solution to path
47: $P_{best}$= getPathFromSolution( $bestSolu$,$PM$ )
48: $\tau$= $bestSoluFit$
49: return ($P_{best}$,$\tau$)
50: end
```

Calculate the Sub-path Matrix after Avoiding NFZs

The pseudocode used for calculate the sub-path matrix is in Algorithm 2. As seen in the Algorithm 2, in steps 6 to 16, it is checked whether the line segment composed of any two points in *VAS* has an intersection with the line segment defined by *NFZS*. If there is no intersection, the corresponding element value in the sub-path matrix is assigned to the sub-path composed of these two points. If there is an intersection, it means that the line segment passes through the NFZs, then start the NFZ avoidance algorithm to obtain the sub-path and assign the obtained sub-path to the element in the sub-path matrix. The procedure GetSubpathByFloyd in step 18 will be described in later subsection.

For example, in the CPP model instance shown in Fig. 1, the line segment consisting of V_{A6} and V_{A7} has no intersection with any line segment in *NFZS*, so the value of $PM[6][7]$ is $SP(V_{A6}, V_{A7})$. And the line segment $LS(V_{A7}, V_{A8})$ has an intersection with the line segment $LS(V_{N0}, V_{N5})$ in *NFZS*, so the value of $PM[7][8]$ is $SP(V_{A7}, V_{N0}, V_{N1}, V_{A8})$ after NFZ avoidance processing.

Algorithm 2 The pseudocode of calculating the sub-path matrix avoiding NFZs

```
    Input : The CPP instance;
      $G$: Instance of CPP model
    Output : The sub-path matrix;
      $PM$: The matrix of sub-path between any two points in VAS after avoiding
    NFZs
 1: begin
    // Get the count of A-Type points from the CPP model instance
 2: $VANum$ = getCount($VAS$)
    // Get the count of line segments from the NFZS
 3: $NFZSNum$ =getCount($NFZS$)
 4: for $i$=1 to $VANum$ do
 5:   for $j$=1 to $VANum$ do
 6:     $V_{Ai} = VSA[i]$
 7:     $V_{Aj} = VSA[j]$
 8:     $LS_x = (V_{Ai}, V_{Aj})$
 9:     $passNFZ$ = false
      // Check if $LS_x$ passes through the NFZs
10:     for $k$=1 to $NFZSNum$ do
11:       $LS_y = NFZS[k]$
12:       if $LS_x \cap LS_y \neq \phi$ then
13:         $passNFZ$ = true
14:         break
15:       end if
16:     end for
17:     if $passNFZ$ then
          // Start NFZ avoidance algorithm to obtain the sub-path after NFZ
    avoidance
18:       $SP_{ij} = GetSubpathByFloyd(VNS, LS_x)$
19:     else
20:       $SP_{ij} = (V_{Ai}, V_{Aj})$
21:     end if
22:     $PM[i][j] = SP_{ij}$
23:   end for
24: end for
25: return ($PM$)
26: end
```

The Procedures for Solution

As seen in the Algorithm 1, the main processing flow for the solution archive in the PSAACO algorithm is indicated below:

- The first process applies the "InitializeSolutionArchive" procedure in step 10 to initialize the solution archive, and uses "UpdateSolutionArchive" procedure in step 11 to update the archive.
- The second process corresponds to steps 21 to 23 in the algorithm. Each ant selects a base solution from the solution archive, constructs one new solution from the base solution by "ConstructSolution" procedure, and then adds the new solution into the solution archive.
- The third process uses the "UpdateSolutionArchive" procedure to sort the solutions in the solution archive by their quality, then keep the best n solutions in the solution archive and remove redundant solutions.
- Repeat the second and third process before a termination condition is satisfied.

The structure of solution archive The structure of solution archive S of PSAACO algorithm is shown in Fig. 2. The solution archive stores n solutions. Each solution s_j is represented by an m-dimensional vector, as shown in the following formulas:

$$s_j = (s_j^1, s_j^2, \cdots, s_j^i, \cdots, s_j^m) \tag{11}$$

$f(s_j)$ and ω_j means the fitness value and selection weight of solution s_j ,respectively.

For the CPP problem, each solution of PSAACO corresponds to a planned path without NFZ avoidance, and each variable in the solution corresponds to the *sn* of one point in the path. For example, the solution s_i corresponding to the path P_i described in Eq. 1 is as follows:

$$s_i = s(4, 3, 2, 1, 0, 5, 6, 7, 8, 11, 10, 9, 12, 13, 14) \tag{12}$$

Initializing solution archive The pseudocode used for generating the initial solutions for the solution archive is presented in Algorithm 3. As seen in the algorithm, each solution was randomly generated as a permutation of the set of the points in *VSA*. Then, each solution is converted to a path using the "getPathFromSolution" procedure in step 4. The fitness value of the path is calculated in step 5, and the new solution is added into the solution archive in step 6.

Algorithm 3 The pseudocode to initialize the solution archive

```
Input : The CPP instance and the sub-path matrix;
    $G$: Instance of CPP model, $PM$: The sub-path matrix,
    $\psi$: UAV rotation rate, $\nu$: UAV speed,
    $n$: Number of solutions in solution Archive
Output : The solution archive;
    $S$: The solution archive
1: begin
2: for $i$=1 to $n \times 2$ do
      // The random generation of each solution as the permutation of the set of the points in $VSA$
3:    $newSolution$ = randperm($VSA$)
      // Convert solution to path
4:    $P_i$ = getPathFromSolution($newSolution$ , $PM$)
      // Calculate the fitness value of the path
5:    $newSolutionFit$ = calculateFitnessValue($P_i$, $\psi$, $\nu$)
      // Add the new solution into the solution archive
6:    AddSolutionIntoArchive ($newSolution$, $newSolutionFit$)
7: end for
8: return ($S$)
9: end
```

The pseudocode of the "getPathFromSolution" procedure is shown in Algorithm 4. Take out two adjacent points of the solution *solu* in turn, query their corresponding sub-paths from *PM*, and combine these sub-paths into a complete path *P*. Since the sub-paths in *PM* are NFZs-avoiding, the new path is also NFZs-avoiding.

Algorithm 4 The pseudocode to convert a solution into a path

```
Input : A base solution and the sub-path matrix;
    $solu$: A base solution,
    $PM$: The matrix of sub-path between any two points in VAS after avoiding NFZs;
Output : A path from the solution
    $P$: A path
1: begin
   // Get the dimension of a solution
2: $soluDim$ = getDimension ($solu$)
3: for $i$=2 to $soluDim$ do
4:    $x = solu[i-1]$
5:    $y = solu[i]$
6:    $SP_{ij} = PM[x][y]$
7:    AddSubpathIntoSolution ($P$, $SP_{ij}$)
8: end for
9: return ($P$)
10: end
```

Constructing new solutions probabilistically First, each ant applies the roulette wheel selection algorithm, probabilistically selecting a solution from the solution archive S. As shown in the following equation, the selection probability PS_j of the solution s_j is calculated according to its selection weight ω_j.

$$PS_j = \frac{\omega_j}{\sum_{i=1}^{n} \omega_i} \tag{13}$$

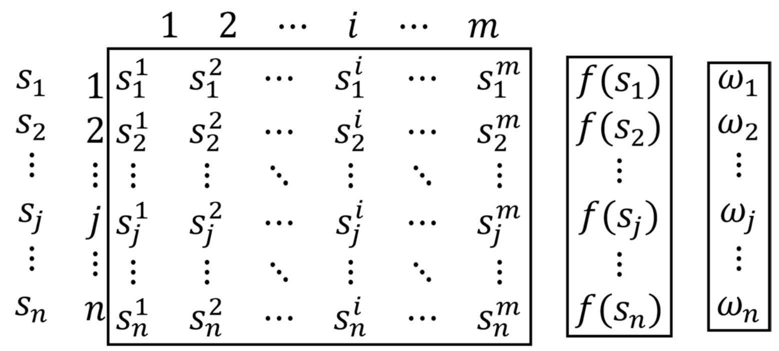


Fig. 2 The structure of solution archive

Where ω_j can be calculated by Eq. 14. In the equation, η represents the search size constant, and n is the number of solutions in S.

$$\omega_j = \frac{1}{\eta n \sqrt{2\pi}} e^{\frac{-(j-1)^2}{2\eta^2 n^2}} \tag{14}$$

Then, a new solution is built based on the selected solution. Similarly to [58], this work uses the local search operators to construct a solution. The pseudocode to construct a new solution is shown in Algorithm 5. We use the inversion and insertion operators, which are among the most popular local search operators. The algorithm chooses to use the inversion operator with a probability of 50%, as shown in steps 6 and 7. And the insertion operator is selected with a probability of 25% in steps 8 and 9. In steps 10 to 22, we apply the inversion operator multiple times in a continuous loop, and take the best constructed solution as the new solution. The local search operators can be described as follows:

- **The inversion operator**: The inversion operator simply inverses the variables of the solution between positions x and y. In other words, it randomly selects two positions on the solution and then reverses the sub-path between these two positions. For example, inversion(s_i, x, y) generates a new solution s_j, which is $s_j = (s_i^1, \cdots, s_i^{x-1}, s_i^y, s_i^{y-1}, \cdots, s_i^x, s_i^{y+1}, \cdots, s_i^n)$, where $1 \leq x < y \leq n$.
- **The insertion operator**: The insertion operator randomly selects the positions x and y in the solution. Then it moves the variable from Position y to Position x. Insertion(s_i, x, y) generates a new solution s_j, which is $s_j = (s_i^1, \cdots, s_i^{x-1}, s_i^y, s_i^x, s_i^{x+1}, \cdots, s_i^{y-1}, s_i^{y+1}, \cdots, s_i^n)$, where $1 \leq x < y \leq n$.

Algorithm 5 The pseudocode to construct a new solution from a base solution

```
Input : A base solution and the sub-path matrix;
    $solu$: A base solution,
    $PM$: The matrix of sub-path between any two points in VAS after avoiding NFZs,
    $\psi$: UAV rotation rate, $\nu$: UAV speed,
Output :A new solution and its fitness value
    $newSolution$: A new solution,
    $newSolutionFit$: The fitness value of the new solution
1: begin
2: $soluDim$ = getDimension ($solu$)
    // Generate a random number between 0 and 100
3: $r1$ = random(0, 100)
4: $r2$ = random(0, soluDim)
5: $r3$ = random(0, soluDim)
6: if $r1 < 50$ then
7:     $newSolution$=ApplyInversionOperator($solu, r2, r3$)
8: else if $r1 < 75$ then
9:     $newSolution$=ApplyInsertOperator($solu, r2, r3$)
10: else
11:     $rmin$ = getMin($r2, r3$)
12:     $rmax$ = getMax($r2, r3$)
13:     $minfit = 0$
14:     for $i$=$rmin$ to $rmax$ do
15:         $solu$ = ApplyInversionOperator($solu, i, i+1$)
16:         $P_i$= getPathFromSolution($solu, PM$)
17:         $fit_i$ = calculateFitnessValue($P_i, \psi, \nu$)
18:         if $minfit == 0$ or $minfit > fit_i$ then
19:             $newSolution$= $solu$
20:             $minfit = fit_i$
21:         end if
22:     end for
23: end if
24: $P$ = getPathFromSolution($newSolution, PM$)
25: $newSolutionFit$ = calculateFitnessValue($P, \psi, \nu$)
26: return ($newSolution, newSolutionFit$)
27: end
```

Updating the solution archive The "InitializeSolutionArchive" and "ConstructSolution" procedures add some new solutions into the solution archive S. The "UpdateSolutionArchive" procedure performs the update S operation. First, the solutions in S containing new solutions are sorted according to their fitness function value, and the solutions with good quality are ranked above the solutions with poor quality. Then, the top n solutions with the best quality in S are retained, and other redundant solutions are removed.

The Procedures for Self-Adptive Parameters

Some parameters greatly affect the performance of PSAACO. The scheme of manually setting parameters depends too much on the experience of setting personnel. This paper adopts a self-adaptive parameters setting scheme for PSAACO.

The self-adaptive parameter archive C is added to store these self-adaptive parameters. Assume that there are a parameter setting schemes, and each scheme is a vector composed of b self-adaptive parameters, the corresponding parameter archive is shown in Fig. 3. c_j^i is the i-th parameter value in the j-th parameter vector. $f(c_j)$ is the fitness value of the c_j. ω_j means the weight of c_j.

In the algorithm 1, the main processing flow for the self-adaptive parameters in the PSAACO algorithm is indicated below:

- The first process applies the "InitializelParameterArchive" procedure in step 7 to initialize the parameter archive. Randomly create a parameter vectors and store them into the parameter archive.
- The second process corresponds to steps 15 to 29 in the algorithm. Each iteration selects a base parameter vector from the archive, constructs a new parameter vector from the base parameter vector by "ConstructParameter" procedure. Take this new parameter vector as the current parameter vector for this iterative calculation. Record the fitness value of the best solution constructed by ants in this iteration, and take this value as the fitness value of the new parameter vector. Add the new parameter vector and its fitness value into the parameter archive.
- The third process uses the "UpdateParameterArchive" procedure to sort the parameter vectors in C by their quality, then keep the best a parameter vectors in C and remove redundant parameter vectors.
- Repeat the second and third process before a termination condition is satisfied.

The procedures for parameter archive are very similar to that for solution archive, and they have almost the same operations in initializing archive, selecting base vector from archive, updating archive, etc.

There is a big different procedure for parameter archive, and that is constructing a new parameter. Each variable in the base parameter vector probabilistically selects new value in its neighborhood, and these new values form a new parameter vector. The probability density function $Pd(x)$ for each variable is as follows:

$$Pd(x) = f(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{15}$$

and

$$\mu = c_j^i, \sigma = \xi \sum_{r=1}^{a} \frac{|c_r^i - c_j^i|}{a-1} \tag{16}$$

$f(x, \mu, \sigma)$ is the Gaussian function, x is a variable in one parameter vector. σ denotes the mean square error , μ is the mean value , and ξ means the algorithm parameter.

Parallel Computing with Multiple Threads

Parallel computing can enhance the algorithm speed, and it introduces parallel computing elements to expand the search mode, which can often improve the quality of algorithm results [59]. In this work, we use a multi-threaded parallel computing approach to improve the solution quality and search speed of PSAACO.

Step 6 of Algorithm 1 starts *TdNum* parallel threads, and steps 7 to 45 in Algorithm 1 are the running steps of each

$$\begin{array}{cc|cccccc|c|c|}
 & & 1 & 2 & \cdots & i & \cdots & b & \multicolumn{1}{c}{} & \multicolumn{1}{c}{} \\
\cline{3-8}
c_1 & 1 & c_1^1 & c_1^2 & \cdots & c_1^i & \cdots & c_1^b & f(c_1) & \omega_1 \\
c_2 & 2 & c_2^1 & c_2^2 & \cdots & c_2^i & \cdots & c_2^b & f(c_2) & \omega_2 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \vdots \\
c_j & j & c_j^1 & c_j^2 & \cdots & c_j^i & \cdots & c_j^b & f(c_j) & \omega_j \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \vdots \\
c_a & a & c_a^1 & c_a^2 & \cdots & c_a^i & \cdots & c_a^b & f(c_a) & \omega_a \\
\cline{3-8}
\end{array}$$

Fig. 3 The structure of parameter archive

thread. Each thread runs an independent ACO algorithm, and it has its own solution archives, parameter archives and global parameters, etc. These threads have independent ant colonies to explore new solutions. And they independently update their own solution archives and parameter archives. After all threads run, take the best solution obtained by all threads as the result of the whole algorithm.

The application of parallel computing speeds up the run of the algorithm, reduces the probability of PSAACO falling into a local optimum, and improves the overall performance of PSAACO. The relevant proof experiments are described in the later experimental part.

The Fitness Value

The most common performance metrics are: path length, number of turning maneuvers, completion time, energy consumption, and quality of area coverage. Path length and turning angle are the two main basic factors, and most common performance metrics usually rely on one of these factors or a combination of two factors. The completion time and energy consumption metrics include the evaluation of these two factors, so these two metrics are more comprehensive and reasonable. In this paper, the completion time is used as the fitness function value.

NFZ Avoidance Using DFA

The Floyd Warshall algorithm is used to search for the shortest path between any two points, and can handle the shortest path problem of undirected graphs or weighted directed graphs [60, 61]. The idea of the algorithm is to insert different intermediate vertices between two points and find out the insertion scheme of the shortest path. Let d_{ij}^x be the shortest path from vertex i to vertex j , and its intermediate vertex is in the set 1, 2... y, where x is the iteration number and y is the total number of all vertices. Then for $y > 1$,

$$d_{ij}^x = min(d_{ij}^{x-1}, d_{ik}^{x-1} + d_{kj}^{x-1}) \tag{17}$$

Thus, d_{ij}^y is the shortest paths matrix calculated by the Floyd Warshall algorithm from the input graph. The Floyd Warshall algorithm cannot dynamically add points. When a new vertex is added into the graph, all shortest path needs to be recalculated, and the time complexity is $O(n^3)$.

This paper improves the algorithm so that when vertices are added, only the changes caused by these vertices are calculated, and the algorithm are named DFA.

The DFA is presented in Algorithm 6. The algorithm has three input parameters. *DistanceM*1 represents the distance matrix that has been calculated so far, and *DistanceM*1[*i*][*j*] records the shortest distance from vertex i to vertex j. *PathM*1 indicates the path matrix that has been calculated, and *PathM*1[*i*][*j*] records the intermediate vertex for the shortest path from vertex i to vertex j. *DistanceA* is an array of distances between new point and calculated points, where *DistanceA*[*i*] records the distance between new point and vertex i. In steps 3 to 17 in Algorithm 6, initialize the *DistanceM*2 and *PathM*2 by copying the calculated data in *DistanceM*1 and *PathM*1 , and adding the data of the new point from *DistanceA*. The steps 18 to 23 of the algorithm calculate and update all paths ending at the new point, and the steps 24 to 29 calculate and update all paths starting at the new point, while the steps 30 to 35 calculate and update all paths with the new point as intermediate point. The calculation and update operations are implemented by the "updateMatrix" procedure. The detailed steps of the "updateMatrix" procedure are described in algorithm 7, and its principle is shown in Formula 17.

It is easy to see that the time complexity of DFA is $O(3n^2)$. When the number of points is large, DFA has great advantages in running speed.

Algorithm 6 The pseudocode to DFA

```
Input : Distance and path matrix for calculated points, array of distances
between new point and calculated points;
  $DistanceM1$: Distance matrix for calculated points,
  $PathM1$: Path matrix for calculated points,
  $DistanceA$: Array of distances between new point and calculated points
Output : Distance and path matrix after calculating the new point;
  $DistanceM2$: Distance matrix after calculating the new point,
  $PathM2$: Path matrix after calculating the new point
1: begin
2: $len$ = getRowLength ($DistanceM1$)
   //Initialize the $DistanceM2$ and $PathM2$
3: for $i = 1$ to $len + 1$ do
4:    for $j = 1$ to $len + 1$ do
5:       if $i \le len$ and $j \le len$ then
6:          $DistanceM2[i][j] = DistanceM1[i][j]$
7:          $PathM2[i][j] = PathM1[i][j]$
8:       else
9:          if $i == len + 1$ then
10:            $DistanceM2[i][j] = DistanceA[j]$
11:         else
12:            $DistanceM2[i][j] = DistanceA[i]$
13:         end if
14:         $PathM2[i][j] = j$
15:      end if
16:   end for
17: end for
   // Calculate and update all paths ending at the new point
18: for $k = 1$ to $len$ do
19:    for $i = 1$ to $len$ do
20:       $j = len + 1$
21:       updateMatrix($DistanceM2, PathM2, i, k, j$)
22:    end for
23: end for
   // Calculate and update all paths starting at the new point
24: for $k = 1$ to $len$ do
25:    $i = len + 1$
26:    for $j = 1$ to $len + 1$ do
27:       updateMatrix($DistanceM2, PathM2, i, k, j$)
28:    end for
29: end for
   // Calculate and update all paths with the new point as intermediate point
30: $k = len + 1$
31: for $i = 1$ to $len + 1$ do
32:    for $j = 1$ to $len + 1$ do
33:       updateMatrix($DistanceM2, PathM2, i, k, j$)
34:    end for
35: end for
36: return ($DistanceM2, PathM2$)
37: end
```

Algorithm 7 The pseudocode to the "updateMatrix" procedure

```
Input : Distance and path matrix, starting point, end point and intermediate point of the path;
    $DistanceM$: Distance matrix, $PathM$: Path matrix,
    $i$: Starting point of the path, $j$: End point of the path,
    $k$: Intermediate point of the path
Output : The updated distance and path matrix;
    $DistanceM$: The updated distance matrix,
    $PathM$: The updated path matrix
1: begin
2: if $i \neq j$ and $DistanceM[i][k] \neq \infty$ and $DistanceM[k][j] \neq \infty$ then
3:     $x = DistanceM[i][k] + DistanceM[k][j]$
4:     if $x < DistanceM[i][j]$ or $DistanceM[i][j] == \infty$ then
5:         $DistanceM[i][j] = x$
6:         $PathM[i][j] = k$
7:     end if
8: end if
9: return $(DistanceM, PathM)$
10: end
```

We apply the DFA to deal with NFZ avoidance. Assume we get a CPP model G and a sub-path $P_k = (V_{Ai}, V_{Aj})$, then the main process for P_k to avoid NFZs is as follows:

- The first process is to calculate all N-Type points in G using DFA, and store the result in the distance matrix $DistanceM1$ and the path matrix $PathM1$. We take the CPP model shown in Fig. 1 as an example, The $DistanceM1$ is shown in Table 2. The data in the table represents the shortest distance between two N-Types points. In particular, ∞ indicates that there is no feasible path between two points. Table 3 describes the data of $PathM1$. From the table we can see that the shortest path from V_{N1} to V_{N10} is: $(V_{N1}, V_{N5}, V_{N10})$, since 5 is the intermediate vertex between V_{N1} to V_{N10}.
- The second process is to calculate the distance between the starting vertex V_{Ai} and all N-Type points to obtain the distance array of $DistanceA1$. For example, the $DistanceA1$ for vertex V_{A7} in Fig. 1 is $(14.1, \infty, \cdots, \infty, 51.0)$.
- The third process is to take $DistanceA1$, $DistanceM1$ and $PathM1$ as input parameters, apply the DFA to calculate and update the distance matrix $DistanceM1$ and the path matrix $PathM1$.
- Repeat process 2 and process 3 to calculate the end vertex V_{Aj}. And we can get the NFZ avoidance path from final path matrix $PathM1$. In the example, the final $DistanceM1$ is shown in Table 4 and the final $PathM1$ is shown in Table 5. It can be seen from the tables that the NFZ avoidance path of $P_k = (V_{A7}, V_{A8})$ is $(V_{A7}, V_{N0}, V_{N1}, V_{A8})$.

Experiment Results and Discussions

In this section, we present experimental results and discussions of our work. In the following, two metrics for evaluating performance are introduced first. Next the performance improvement using self-adaptive parameters and parallel computing is presented. Finally the performance comparison with other algorithms and existing works is carried out.

Experimental Environment

The computer used for the experiments is Lenovo Xiaoxin Pro16 2021: AMD Ryzen 7-5800H CPU (8cores) , 16 GB memory, 1 TB SSD. The integrated development environment of the experiments is Visual Studio 2019 on Windows 11, and the experimental projects are programmed using C#.

Performance Metrics

Performance metrics, completion time and energy consumption are used to evaluate the experimental results. Their definitions are as follows.

Completion Time [62] is the total time required to complete the planned path. It can be calculated:

$$\tau = \frac{\beta}{\nu} + \frac{\varphi}{\psi} \tag{18}$$

Where β is total distance of a path, φ denotes total turning angles of a path, ψ is UAV rotation rate, and ν is UAV speed.

Table 2 Distance matrix for N-Type points

	V_{N0}	V_{N1}	$\cdots$	V_{N9}	V_{N10}
V_{N0}	∞	20	$\cdots$	93.0	63.2
V_{N1}	20	∞	$\cdots$	113.0	76.5
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
V_{N9}	93.0	113.0	$\cdots$	∞	40
V_{N10}	63.2	76.5	$\cdots$	40	∞

Table 3 Path matrix for N-Type points

	V_{N0}	V_{N1}	$\cdots$	V_{N9}	V_{N10}
V_{N0}	0	1	$\cdots$	6	10
V_{N1}	0	1	$\cdots$	6	5
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
V_{N9}	6	6	$\cdots$	9	10
V_{N10}	0	5	$\cdots$	9	10

Table 4 Distance matrix after NFZ avoidance

	V_{N0}	V_{N1}	$\cdots$	V_{N9}	V_{N10}	V_{A7}	V_{A8}
V_{N0}	∞	20	$\cdots$	93.0	63.2	14.1	34.1
V_{N1}	20	∞	$\cdots$	113.0	76.5	34.1	14.1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
V_{N9}	93.0	113.0	$\cdots$	∞	40	79.9	110.7
V_{N10}	63.2	76.5	$\cdots$	40	∞	51.0	70.7
V_{A7}	14.1	34.1	$\cdots$	79.9	51.0	∞	48.3
V_{A8}	34.1	14.1	$\cdots$	110.7	70.7	48.3	∞

Table 5 Path matrix after NFZ avoidance

	V_{N0}	V_{N1}	$\cdots$	V_{N9}	V_{N10}	V_{A7}	V_{A8}
V_{N0}	0	1	$\cdots$	6	10	11	1
V_{N1}	0	1	$\cdots$	6	5	0	12
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
V_{N9}	6	6	$\cdots$	9	10	2	5
V_{N10}	0	5	$\cdots$	9	10	11	12
V_{A7}	0	0	$\cdots$	2	10	11	1
V_{A8}	1	1	$\cdots$	5	10	1	12

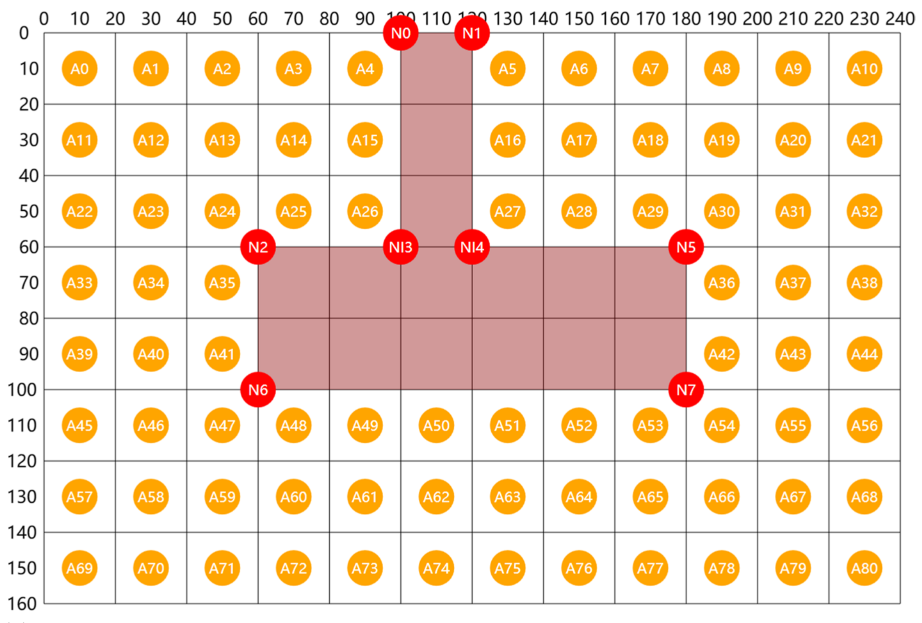


Fig. 4 The CPP model2.[63]

Energy consumption [40] is the total energy consumption required to complete the planned path. It can be calculated:

$$\Xi = \lambda \times \beta + \gamma \times \varphi \tag{19}$$

Where λ is the energy consumption required by the UAV to complete each 1m path length, and γ is the energy consumption required to complete each 1 degree turning angle. In this work, we use the same values as in [40], and

Table 6 Experiments for Performance Improvement using Self-Adaptive Parameters

	Parameters			CPP model1			CPP model2		
	n	***q***	η	**Best**	**Mean**	**Worst**	**Best**	**Mean**	**Worst**
ACO1	5	10	0.001	46	46.31	48	268.64	306.77	397.00
ACO2	10	20	0.01	45.32	46.29	49	242.20	263.62	300.10
ACO3	20	50	0.1	45.32	45.52	46	214.15	227.81	254.74
ACO4	50	100	1	45.32	45.32	45.32	212.65	224.73	244.07
PSAACO	Self-Adaptive			45.32	45.32	45.32	208	224.78	259.54

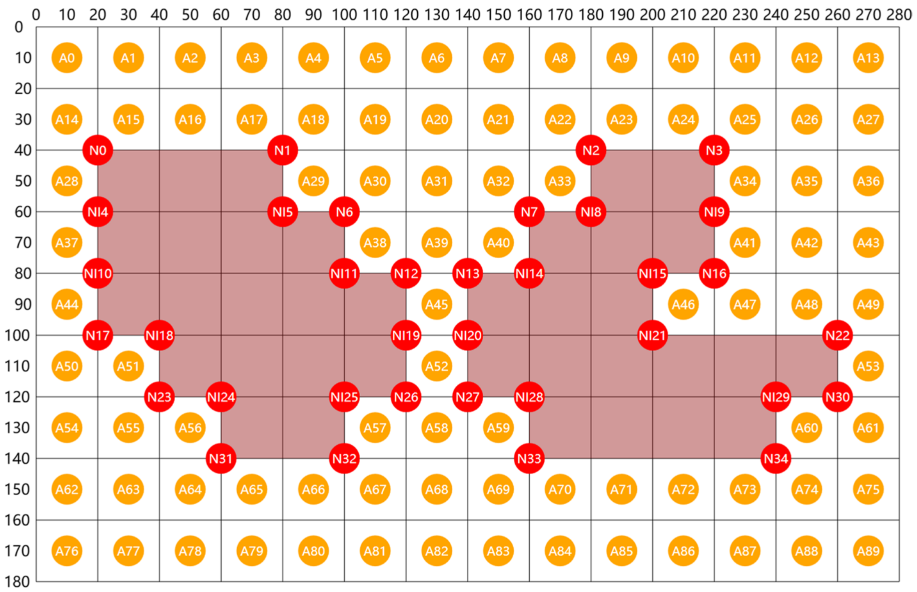


Fig. 5 The CPP model3. [43]

Table 7 Experiments for Performance Improvement using Parallel Computing

Number of parallel threads	CPP model2			CPP model3		
	Best	**Mean**	**Worst**	**Best**	**Mean**	**Worst**
1	208	224.78	259.54	266.30	276.59	295.99
5	208	214.93	236.02	257.79	267.02	285.00
10	208	212.80	223.81	259.60	266.22	273.44
20	208	211.79	226.55	256.72	261.53	266.75

set the values of λ and γ to 0.1164 KJ/m and 0.0173 KJ/degree, respectively.

Performance Improvement using Self-Adaptive Parameters

Some experiments are carried out to test the performance improvement using self-adaptive parameters. Two CPP models are used in these experiments. One CPP model is depicted in Fig. 1, named CPP model1, which has a relatively small area. The other CPP model is from Reference [63] , which has a relatively large area and a complex no-fly zone. We make some modifications, and the adjusted model is named CPP model2, as shown in Fig. 4.

Table 6 lists the parameter setting scheme and performance of five experiments. In the four experiments from ACO1 to ACO4, we manually set the values of the three self-adaptive parameters n, q and η for the ACO

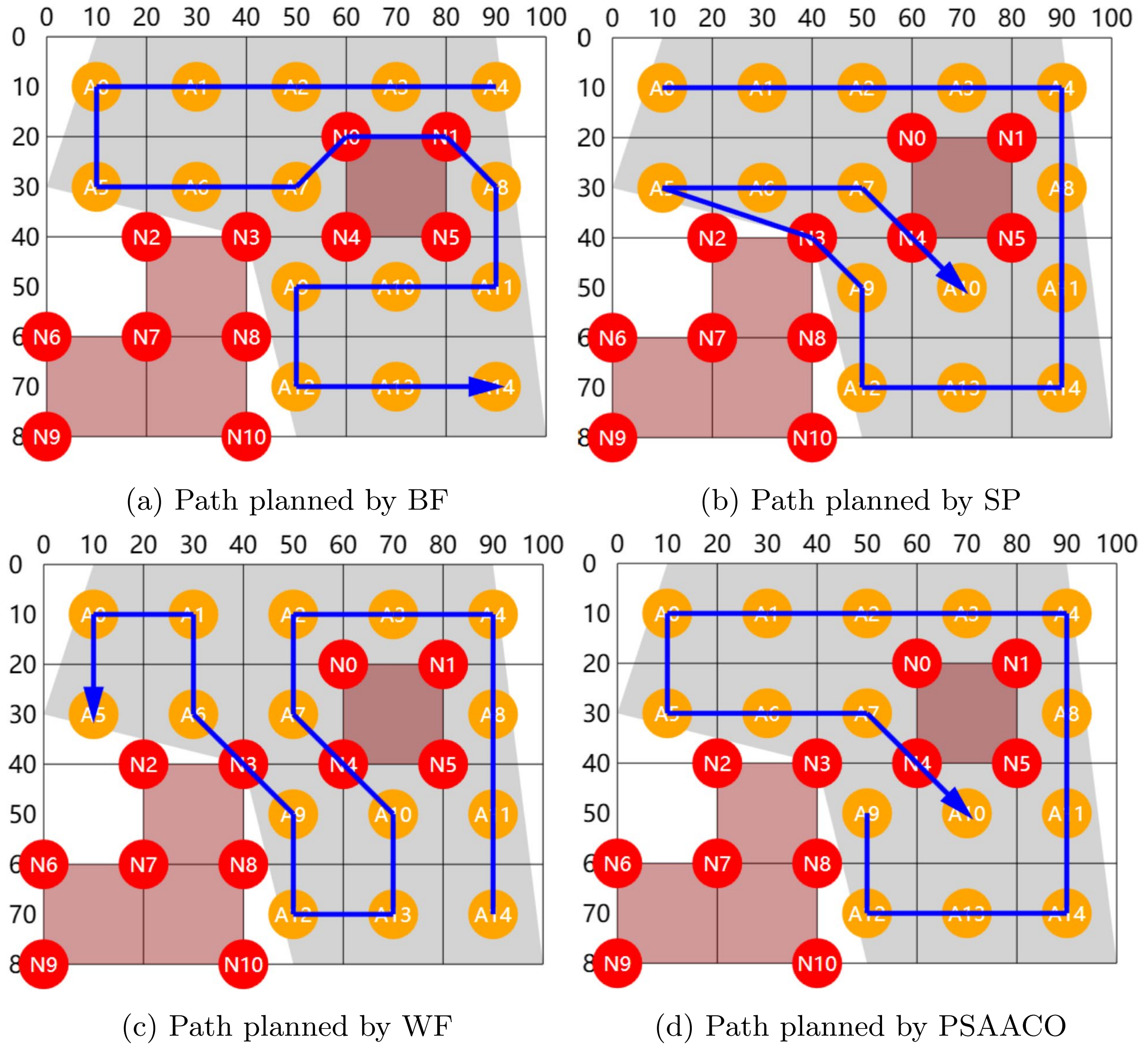
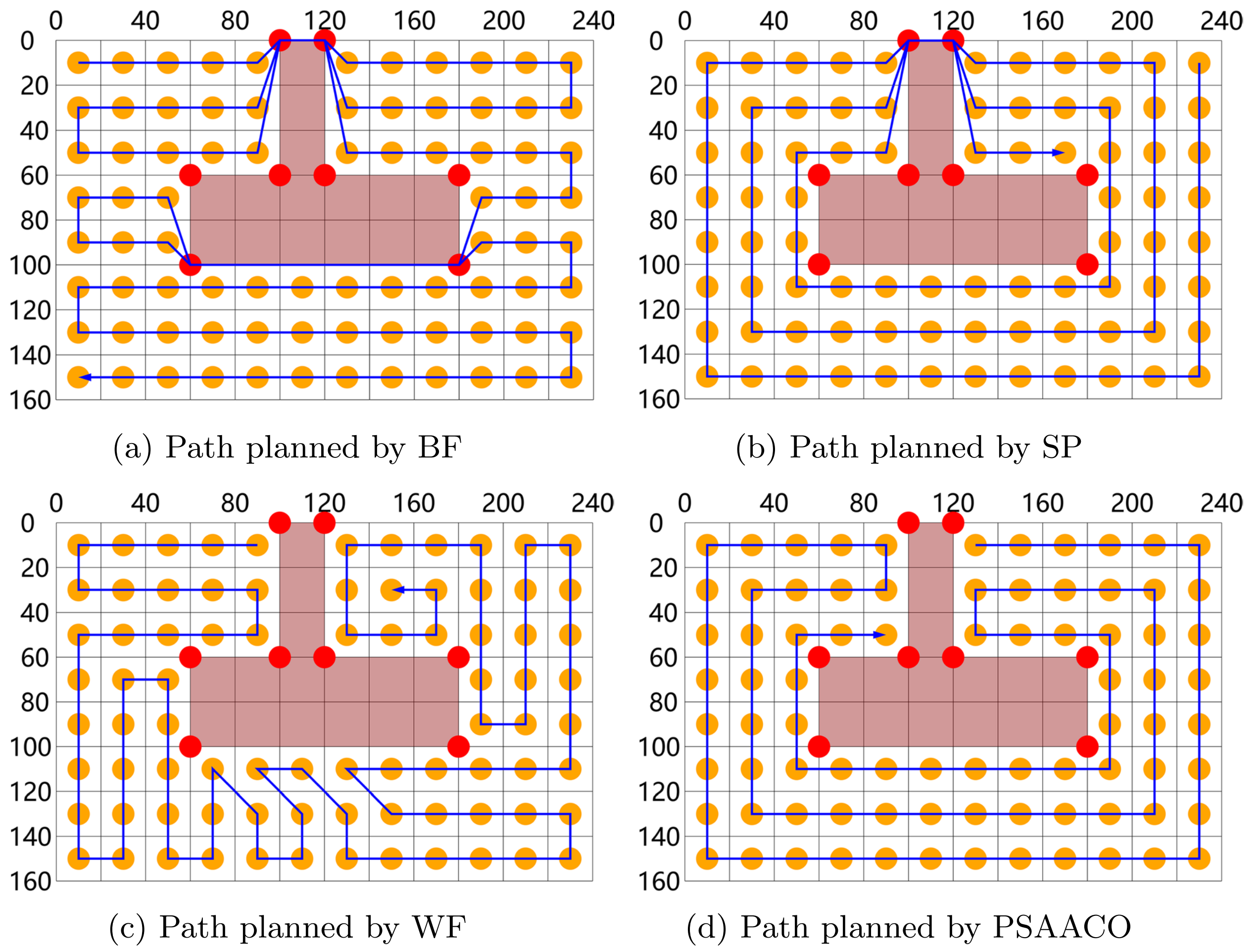


Fig. 6 Performance comparison experiment on CPP model1

algorithm, and the three parameters represent the solution number in solution archive, the ant number for each thread and the search size in archive area, respectively. The fifth experiment shows the experimental result of the PSAACO algorithm. In Table 6, the columns titled "CPP model1" and "CPP model2" show the experimental results of running PSAACO algorithm 10 times on CPP model1 and CPP model2, respectively. The sub column titled "Best" shows the completion time corresponding to the best solution of 10 runs, the sub column titled "Mean" shows the mean completion time, and the sub column titled "Worst" indicates the worst completion time. It can be seen from Table 6:

- When planning the path on CPP model1, most experiments can obtain excellent solutions. This means that parameter settings are less important for small, simple areas. For the large and complex CPP model2, the performance of the third to fifth experiments is significantly better than the other experiments. This means that parameter settings are very important for this model.
- The performance of ACO algorithm largely depends on parameter settings, especially when dealing with some complex models. While PSAACO shows excellent comprehensive performance, even when dealing with complex models.



(a) Path planned by BF (b) Path planned by SP

(c) Path planned by WF (d) Path planned by PSAACO

Fig. 7 Performance comparison experiment on CPP model2

In conclusion, self-adaptive parameter setting can improve the performance of ACO algorithm to solve the CPP problem, and the performance improvement is more obvious when the areas are large or the no-fly zones are complex.

Performance Improvement using Parallel Computing

To test the performance improvement using parallel computing, we conduct some experiments on two CPP models. One CPP model is CPP model2. The other is a modified model based on the model in ref. [43] , which is illustrated in Fig. 5.

Four experiments are done and each experiment runs 10 times. The number of parallel threads for these four experiments is set to 1, 5, 10, and 20, respectively. All experiments adopt self-adaptive parameter setting scheme. Table 7 shows the experimental results. It can be seen from Table 7:

- The performance of PSAACO algorithm is improved as the number of threads increases. As shown in the Table 7, experiments with a large number of threads achieve better mean completion time metrics.
- When the number of threads increases to a certain value, such as more than 10, continuing to increase the number of threads has little effect on performance improvement.

In short, parallel computing can improve the performance of PSAACO algorithm. And a small number of parallel threads can be selected for simple problems, while a larger number of threads, such as 10 or 20, can be selected for complex problems.

Performance Comparison between PSAACO and Other Algorithms

The BF and SP are the most common pattern for a single UAV to explore regular-shaped and non-complex areas

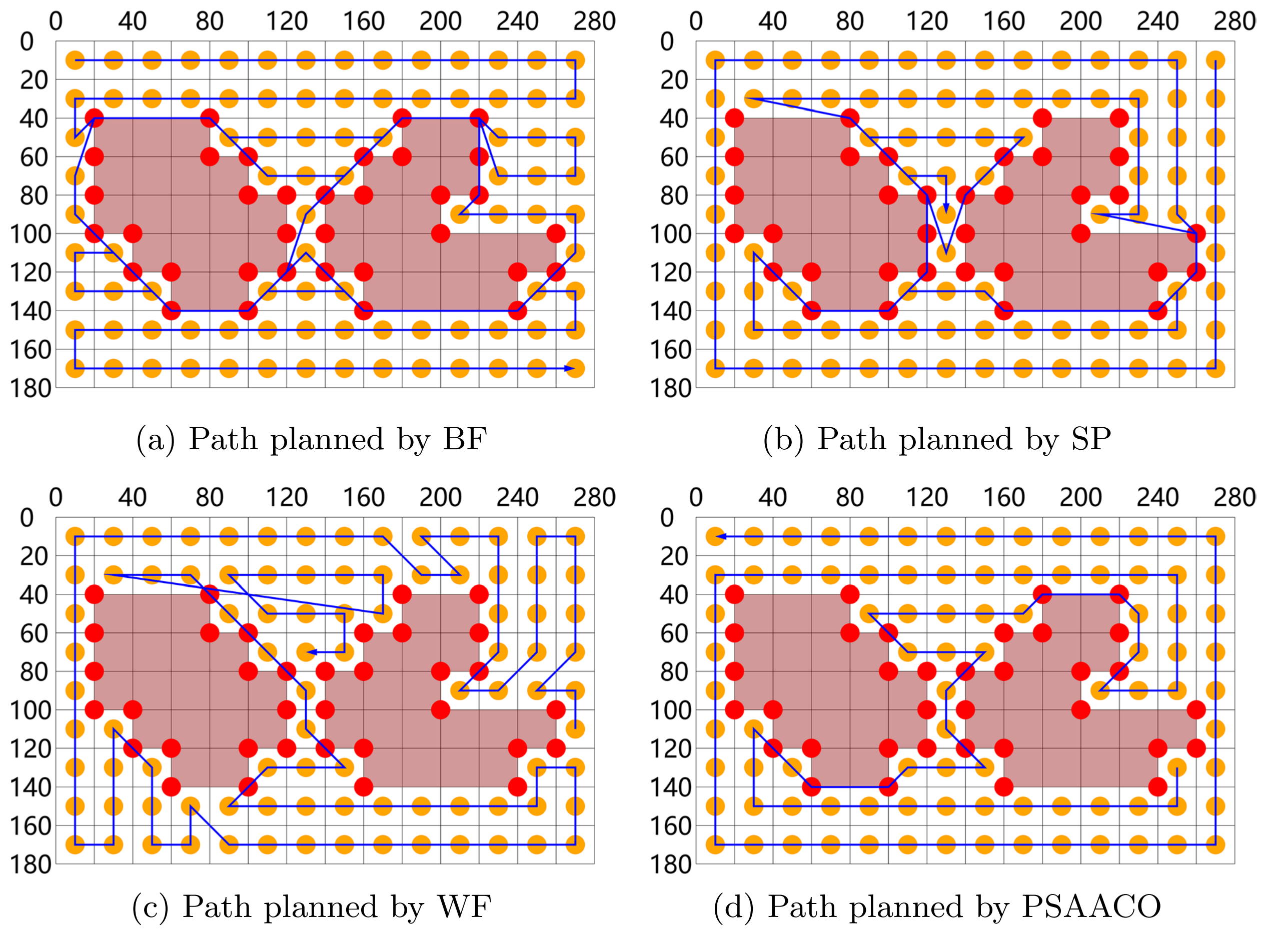


Fig. 8 Performance comparison experiment on CPP model3

Table 8 Experiments for Performance Comparison between PSAACO and Other Algorithms

		BF	**SP**	**WF**	**PSAACO**
CPP model1	Path length[m]	308.20	314.00	296.50	288.20
	Turning angle[deg]	630.00	548.13	720.00	495.00
	Energy consumption[KJ]	46.77	46.03	46.97	42.11
	Completion time[sec]	51.82	49.67	53.65	45.32
CPP model2	Path length[m]	2085.00	1793.50	1633.10	1600.00
	Turning angle[deg]	2507.28	1771.02	2970.00	1440.00
	Energy consumption[KJ]	286.07	239.40	241.47	211.15
	Completion time[sec]	292.08	238.38	262.31	208.00
CPP model3	Path length[m]	2786.40	2472.90	2009.10	1917.90
	Turning angle[deg]	3166.26	3031.02	3510.00	1890.00
	Energy consumption[KJ]	379.11	340.28	294.58	255.94
	Completion time[sec]	384.18	348.32	317.91	254.79

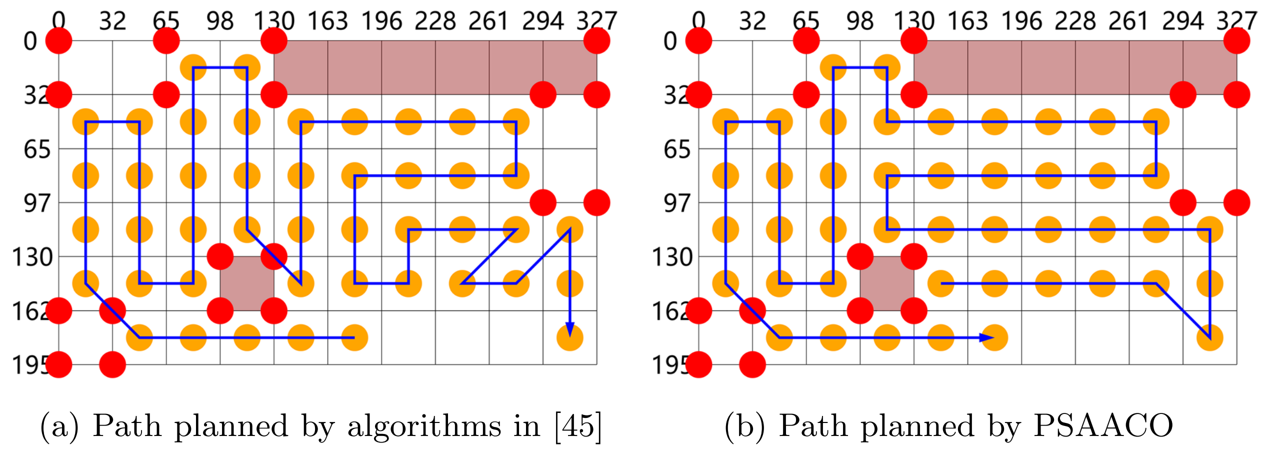

0 32 65 98 130 163 196 228 261 294 327
0 32 65 97 130 162 195
(a) Path planned by algorithms in [45]
(b) Path planned by PSAACO

Fig. 9 Paths planned by the algorithms in [45] and PSAACO

Table 9 Experiments for Performance Comparison between PSAACO and algorithm in [45]

	Ref. [45]	PSAACO
UAV speed[m/sec]	8	8
UAV rotation rate[deg/sec]	30	30
Path length[m]	1488.01	1462.60
Turning angle[deg]	1890	1349.64
Energy consumption[KJ]	205.90	193.59
Completion time[sec]	249.00	227.81

[42]. The wavefront algorithm(WF) is a popular coverage path planning approach suitable for irregular-shaped areas [63]. In this work, we modify these three algorithms to solve the CPP problem as described in Eq. 5.

The main process of this modified BF algorithm is as follows:

- Find the points on the four corners in *VSA*, for example, the four points in Fig. 1 are V_{A0}, V_{A4}, V_{A14}, V_{A12} ,respectively.
- Take each point as the starting point and go back and forth along the horizontal, vertical and diagonal directions to generate paths. For example, Fig. 1e depicts the generated path in the horizontal direction starting from V_{A4}. Each point can generate three paths, and four points can get a total of twelve paths.
- These paths are processed by the DFA for NFZ avoidance, and the fitness values of these paths are calculated. The path with the smallest fitness value is taken as the result of the BF algorithm.

The modification to SP is similar to the modification to BF. First, find the four corner points in the *VSA*. Next, take these points as starting points, and make a spiral motion in a clockwise or counterclockwise direction to generate paths. Then, these paths are processed for NFZ avoidance, and their fitness function values are calculated. Finally, the best path is taken as the result.

In the modified WF algorithm, we select each point in the *VSA* as the goal point, and use the wavefront method and NFZ avoidance to generate paths, then take the best path as the algorithm result.

We do some experiments to compare our algorithm with the three algorithms. CPP model1, CPP model2 and CPP model3 are used in these experiments. CPP model1 has a relatively small area and a simple NFZ, CPP model2 has a relatively large area and a complex NFZ, and CPP model3 has a larger area and a very complex NFZ. The three models cover scenarios of different area and NFZs of different complexity, which can comprehensively test and compare the performance of these algorithms.

Set the number of parallel threads to 20, make 10 path planning for each scenario, and select the optimal solution in the 10 path planning experiments as the final solution of the algorithm. Figure 6a-d represent the CPP on model1 done by BF,SP,WF and PSAACO, respectively. Figures 7 and 8 show the paths planned by the four algorithms on model2 and model3. As can be seen from these figures, PSAACO algorithm has shorter distance and fewer turns than other algorithms.

Table 8 shows that PSAACO generate a shorter path on model1 than BF, SP and WF by 6.5%, 8.2% and 2.8% respectively, decreasing the turning angles by 21.4%, 9.7% and 31.3%, lowering the energy consumption by 10.0%,

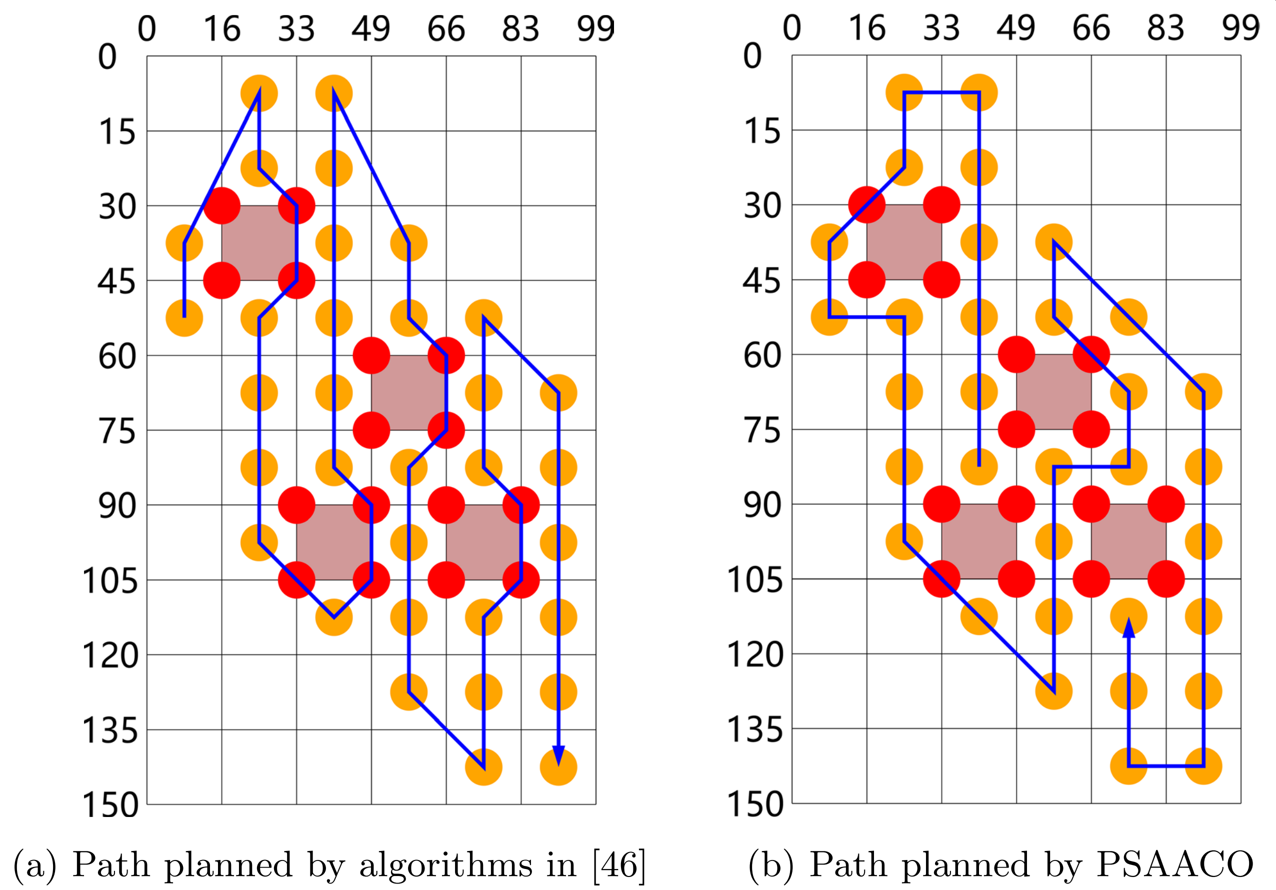

0 16 33 49 66 83 99
0 15 30 45 60 75 90 105 120 135 150
(a) Path planned by algorithms in [46]
(b) Path planned by PSAACO

Fig. 10 Paths planned on area 1 by the algorithms in [46] and PSAACO

Table 10 Experiments on area 1 for performance comparison between PSAACO and algorithm in [46]

	Ref. [46]	PSAACO
UAV speed[m/sec]	8	8
UAV rotation rate[deg/sec]	30	30
Path length[m]	630.55	531.14
Turning angle[deg]	1572.01	1272.00
Energy consumption[KJ]	100.59	83.83
Completion time[sec]	131.22	108.79

8.5% and 10.3%, and reducing the completion time by 12.5%, 8.8% and 15.5%.

Table 8 also shows that the path generated by PSAACO on model2 is 23.3%, 10.8% and 2.0% shorter than BF, SP, and WF, respectively, reduces the rotation angle by 42.6%, 18.7% and 51.5%, decreases the energy consumption by 26.2%, 11.8% and 12.6%, and shortens the completion time by 28.8%, 12.7% and 20.7%.

It can also be seen from the Table 8 that the path generated by PSAACO on model3 is 31.2%, 22.4% and 4.5% shorter than BF, SP and WF, respectively, and the rotation angle is reduced by 40.3%, 37.6% and 46.2%, the energy consumption is reduced by 32.5%, 24.8% and 13.1%, and the completion time is shortened by 33.7%, 26.9% and 19.9%, respectively.

In short, PSAACO has excellent comprehensive performance on CPP. Compared with BF, SP and WF, PSAACO has a great gain in the four indicators of total path length, total turning angle, energy consumption and completion time.

Performance Comparison with Existing Works

In this section, we consider comparing our work with different scenarios in existing works:

- The work in [45], where the deep-limited search is applied to build coverage path.

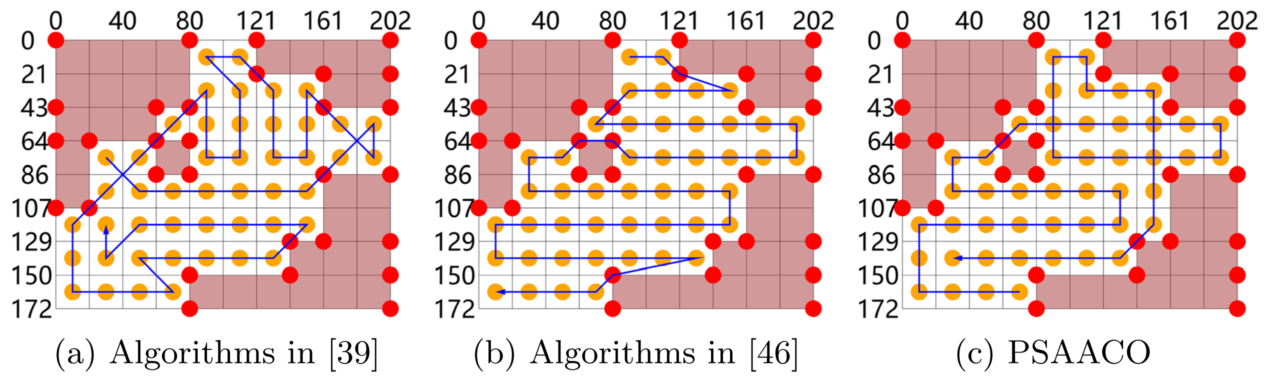


Fig. 11 Paths planned on area 2 by the algorithms in [39, 46] and PSAACO

Table 11 Experiments on area 2 for performance comparison between PSAACO and algorithm in [39, 46]

	Ref. [39]	Ref. [46]	PSAACO
UAV speed[m/sec]	8	8	8
UAV rotation rate[deg/sec]	30	30	30
Path length[m]	1075	1046.08	1025.30
Turning angle[deg]	2030.35	1571.18	1353.57
Energy consumption[KJ]	160.25	148.94	142.76
Completion time[sec]	202.05	183.13	173.28

- An energy-aware grid based algorithm with obstacle avoidance, proposed in [46],
- An grid-based coverage path planning over irregular-shaped area, presented in [39].

The experimental settings are the same as the previous section, we set the number of parallel threads to 20, and choose the optimal solution in 10 runs as the final solution. For a fair comparison of different methods, we assume that the UAV flies with the same speed and rotation rate in the same scenario.

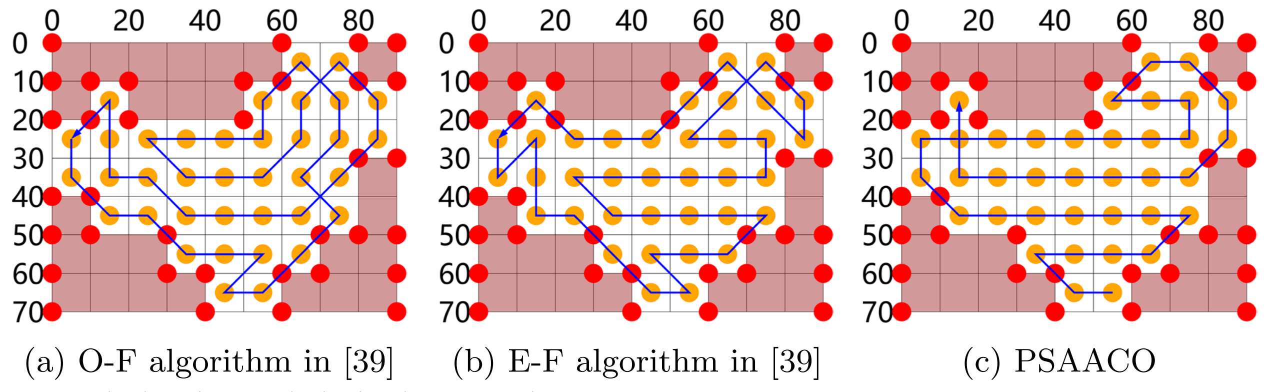


Fig. 12 Paths planned on area 1 by the algorithms in [39] and PSAACO

Table 12 Experiments on area 1 for performance comparison between PSAACO and algorithm in [39]

	O-F algorithm	E-F algorithm	PSAACO
UAV speed[m/sec]	10	10	10
UAV rotation rate[deg/sec]	30	30	30
Path length[m]	440.41	432.13	394.85
Turning angle[deg]	1845	1935	1170
Energy consumption[KJ]	83.18	83.77	66.20
Completion time[sec]	105.54	107.71	78.49

Scenario 1

In this scenario, our approach is compared with that of Valente et al. [45]. The area size of scenario 1 is approximately 327 m × 195 m. In [45], the distance transform function is used over the grid, the deep-limited search is applied to plan coverage paths, and the path with the smallest number of turns is selected.

Figure 9a represents the CPP generated in the [45], Fig. 9b shows the CPP in our work, and Table 9 presents a comparison of the results of the two

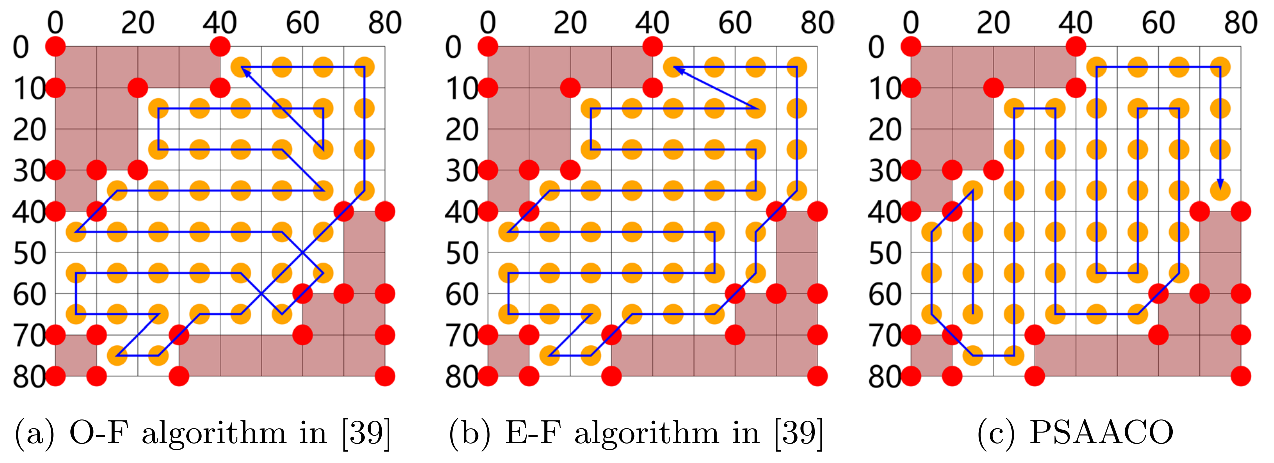


Fig. 13 Paths planned on area 2 by the algorithms in [39] and PSAACO

Table 13 Experiments on area 2 for performance comparison between PSAACO and algorithm in [39]

	O-F algorithm	E-F algorithm	PSAACO
UAV speed[m/sec]	10	10	10
UAV rotation rate[deg/sec]	30	30	30
Path length[m]	509.70	483.07	452.42
Turning angle[deg]	1755	1683.43	1260
Energy consumption[KJ]	89.69	85.35	74.46
Completion time[sec]	109.47	104.42	87.24

approaches.Table 9 shows that the path length obtained by PSAACO is not much reduced compared with that obtained in Reference [45], but the turning angle is greatly reduced, and the angle of PSAACO is reduced by 28.6%. Benefiting from the smaller path length and turning angle, the energy consumption and completion time of PSAACO are improved by 6.0% and 8.5% respectively.

Scenario 2

In this scenario, our approach is compared with the energy-aware grid based solution for obstacle avoidance(EAOA) provided by Ghaddar et al. [46]. Two areas, named area 1 and area 2, in [46] are selected for comparison. In order to fit the CPP model in our work, we make minor adjustments to the paths of these two areas.

Area 1 Figure 10a illustrates the CPP on area 1 generated by EAOA in [46] , and Fig. 10b shows the CPP obtained by PSAACO algorithm.

As can be seen from Table 10, the path length and turning angle of the PSAACO algorithm are greatly improved,

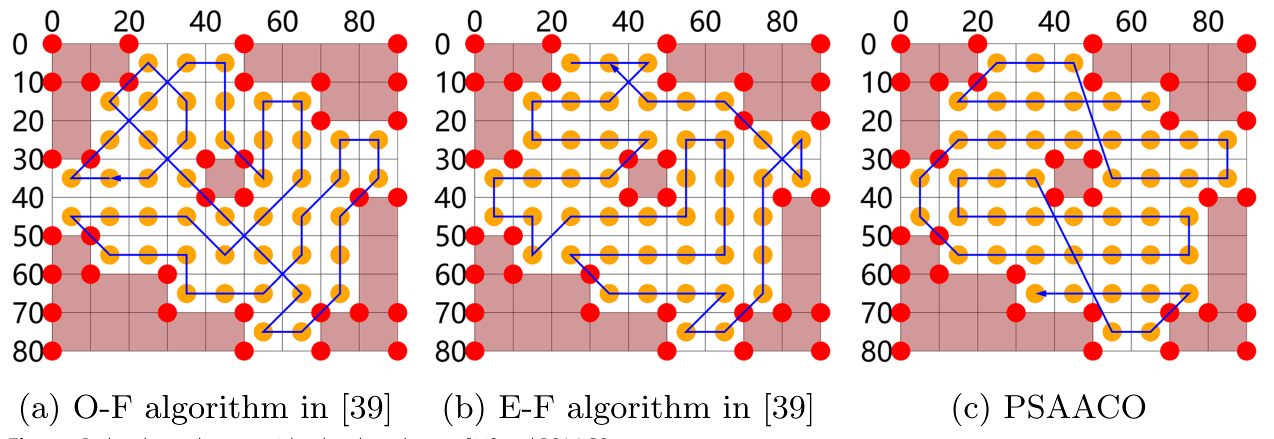


Fig. 14 Paths planned on area 3 by the algorithms in [39] and PSAACO

Table 14 Experiments on area 3 for performance comparison between PSAACO and algorithm in [39]

	O-F algorithm	E-F algorithm	PSAACO
UAV speed[m/sec]	10	10	10
UAV rotation rate[deg/sec]	30	30	30
Path length[m]	556.98	511.42	532.91
Turning angle[deg]	2250	2295	1350
Energy consumption[KJ]	103.75	99.23	85.38
Completion time[sec]	130.69	127.64	98.29

As can be seen from Table 11, the PSAACO algorithm produces a shorter path length and smaller turning angle. Compared with E-F and EAOA, the path length of PSAACO is reduced by 4.6% and 2.0%, and the turning angle is reduced by 33.3% and 13.9%, respectively. Benefiting from the smaller path length and turning angle, the energy consumption of PSAACO is decreased by 14.2% and 5.4%, and the completion time is reduced by 10.9% and 4.1%, respectively.

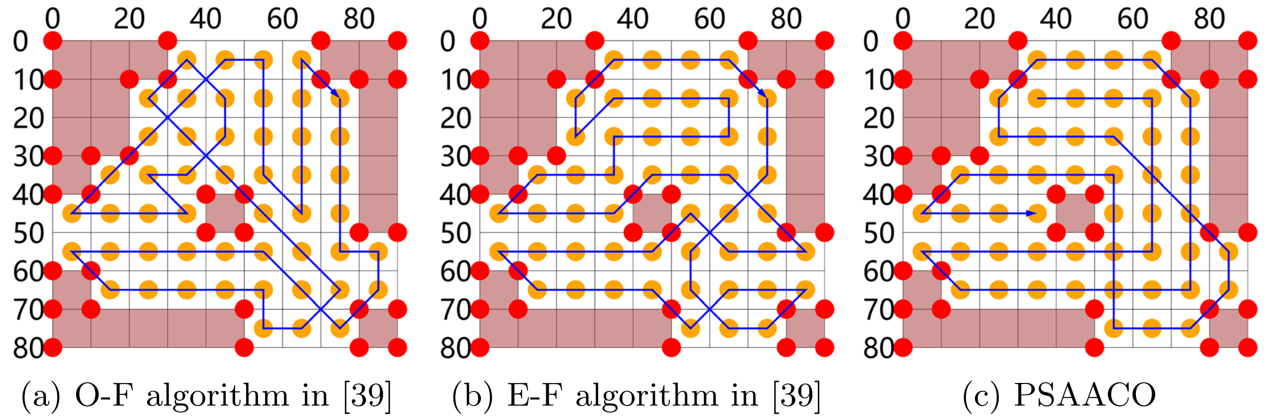


Fig. 15 Paths planned on area 4 by the algorithms in [39] and PSAACO

Table 15 Experiments on area 4 for performance comparison between PSAACO and algorithm in [39]

	O-F algorithm	E-F algorithm	PSAACO
UAV speed[m/sec]	10	10	10
UAV rotation rate[deg/sec]	30	30	30
Path length[m]	582.84	566.27	563.13
Turning angle[deg]	2025	2025	1260
Energy consumption[KJ]	102.87	100.94	87.34
Completion time[sec]	125.78	124.12	98.31

which are reduced by 15.8% and 19.1%, respectively. Correspondingly, energy consumption and completion time have also been greatly improved, reducing by 17.1% and 16.7%, respectively.

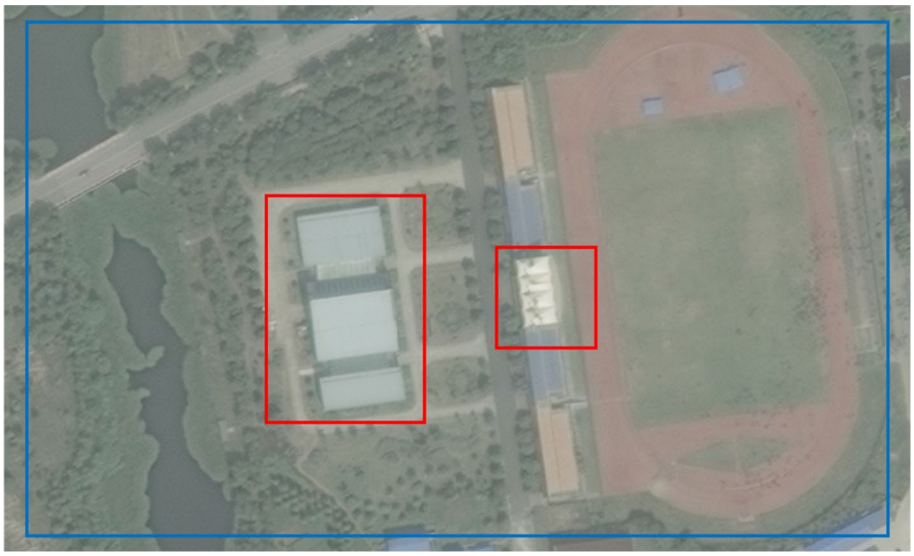
Area 2 Figure 11a shows the CPP on area 2 generated by E-F method in [39], Fig. 11b shows the CPP generated by EAOA in [46], and the CPP obtained in our work is illustrated in Fig. 11c.

Scenario 3

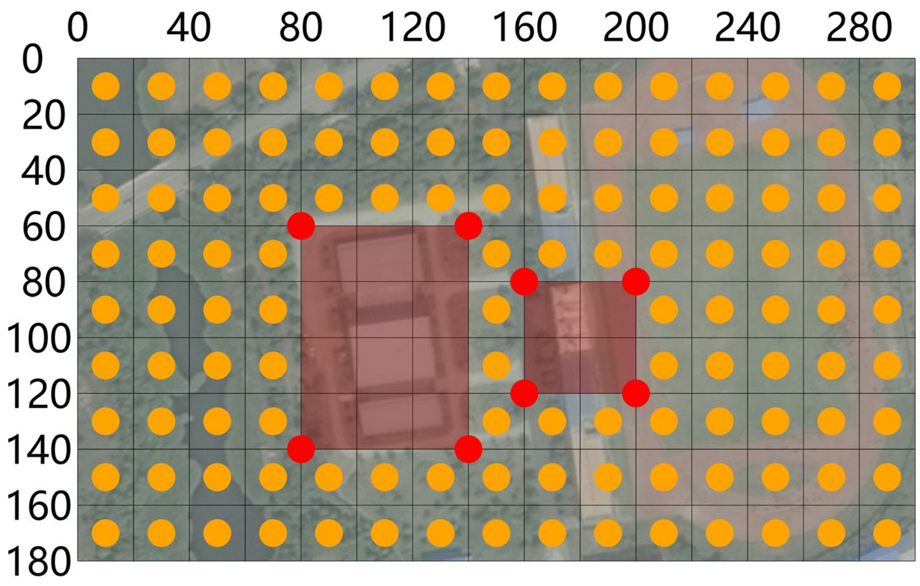
In this scenario, our approach is compared with the algorithm proposed in Cabreira et al. [39]. Four areas, named area 1, area 2, area 3 and area 4, in [39] are selected for comparison. Authors in [39] propose two methods: O-F and E-F, which calculate the path with the lowest turning angle and the path with the least energy consumption respectively. For the convenience of comparison, we modified some parameters, such as the size of some areas, the speed of UAV, etc. But we maintain the same path planning scheme, keeping the number and order of points in the path unchanged.

Area 1 For area 1, the path planned by O-F method is illustrated in Fig. 12a, the path obtained by E-F method is shown in 12b, and the path generated by PSAACO algorithm is presented in 12c.

It can be seen from the Table 12 that the PSAACO algorithm greatly reduces the path length and turning angle. Compared with O-F and E-F methods, the path length of PSAACO is reduced by 10.3% and



(a) Area of interest



(b) Area after decomposing and labeling

Fig. 16 The CPP model4

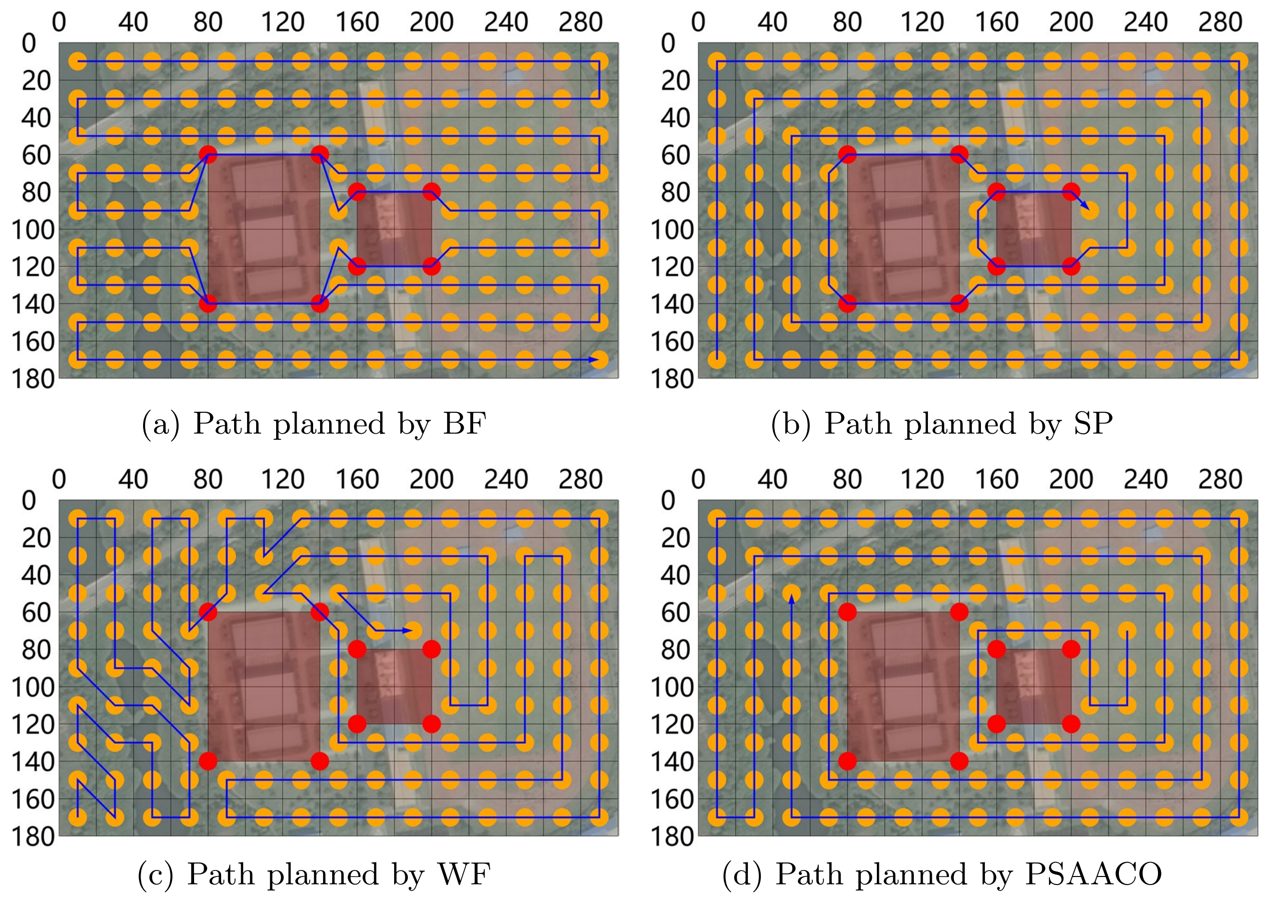


Fig. 17 Paths planned on CPP model4

8.6%, and the turning angle is reduced by 36.6% and 39.5%, respectively. Correspondingly, the energy consumption and completion time of PSAACO have also been greatly improved. The energy consumption of PSAACO is decreased by 20.4% and 21.0%, and the completion time is reduced by 25.6% and 27.1%, respectively.

Area 2 For area 2, the path planned by O-F method is illustrated in Fig. 13a, the path obtained by E-F method is shown in 13b, and the path generated by PSAACO algorithm is presented in 13c.

It can be seen from the Table 13 that the PSAACO algorithm greatly reduces the path length and turning angle. Compared with O-F and E-F methods, the path length of PSAACO is reduced by 11.2% and 6.3%, and the turning angle is reduced by 28.2% and 25.2%, respectively. Correspondingly, the energy consumption and completion time of PSAACO have also been greatly improved. The energy consumption of PSAACO is decreased by 17.0% and 12.8%, and the completion time is reduced by 20.3% and 16.5%, respectively.

Area 3 For area 3, the path planned by O-F method is illustrated in Fig. 14a, the path obtained by E-F method is shown in 14b, and the path generated by PSAACO algorithm is presented in 14c.

As can be seen from Table 14, the path length of PSAACO is 4.3% shorter than the path length in [39] using O-F, but 4.2% longer than the path length using E-F. The PSAACO algorithm greatly reduces the turning angle. Compared with O-F and E-F methods, the turning angle of PSAACO is reduced by 40.0% and 41.2%, respectively. Correspondingly, the energy consumption and completion time of PSAACO have also been greatly improved. The energy consumption of PSAACO is decreased by 17.7% and 14.0%, and the completion time is reduced by 24.8% and 23.0%, respectively.

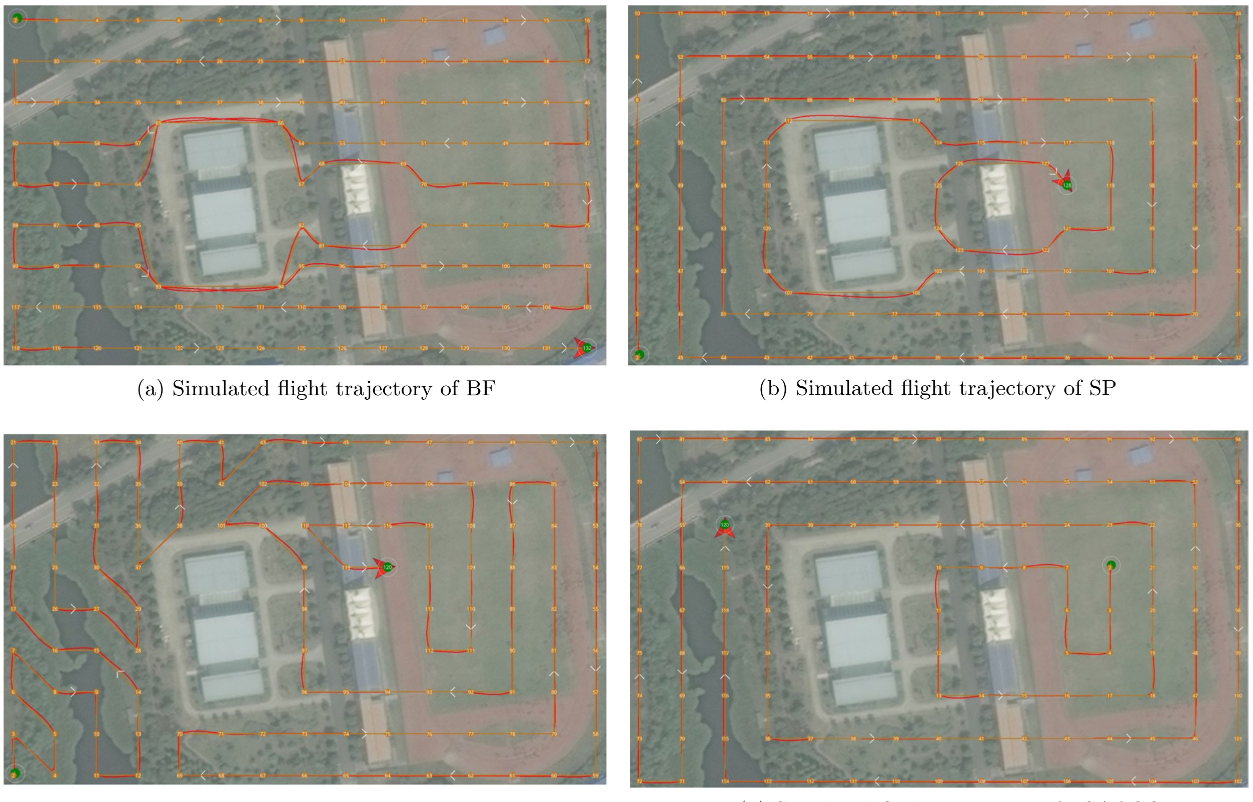

(a) Simulated flight trajectory of BF
(b) Simulated flight trajectory of SP
(c) Simulated flight trajectory of WF
(d) Simulated flight trajectory of PSAACO

Fig. 18 Simulated flight trajectory using Gazebo on CPP model4

Table 16 Simulation experiment results in Gazebo on CPP model4

	BF	SP	WF	PSAACO
Planned completion time[sec]	441.03	385.64	436.42	349
Simulated completion time[sec]	466	414	462	373
completion time difference[sec]	24.97	28.36	25.58	24
Percentage of completion time difference	5.4%	6.9%	5.5%	6.4%

Area 4 For area 4, the path planned by O-F method is illustrated in Fig. 15a, the path obtained by E-F method is shown in 15b, and the path generated by PSAACO algorithm is presented in 15c.

As can be seen from Table 15, the PSAACO algorithm slightly reduces the path length, and the path length of PSAACO is reduced by 3.4% on O-F method and 0.6% on E-F method. The PSAACO algorithm greatly decreases the turning angle, the turning angle of PSAACO is reduced by 37.8% on O-F method and by 37.8% on E-F method. Correspondingly, the energy consumption and completion time of PSAACO have also been greatly improved. The energy consumption of PSAACO is decreased by 15.1% on O-F method and 13.5% on E-F method, and the completion time is reduced by 21.8% on O-F method and 20.8% on E-F method, respectively.

Performance Verification using Gazebo Simulation

In order to verify the feasibility of the proposed algorithm in the real scenarios, some simulation experiments are carried out using a small quadrotor UAV in gazebo. The UAV is equipped with a Pixhawk flight controller running PX4. The ground control station adopts QGroundControl which provides full flight control and mission planning. The communication between the UAV and the ground control station adopts MAVLink protocol.

The simulation experiments use a real scenario with a rectangular area of 300 m × 180 m. There are two tall buildings in this scenario, which are set as NFZs. The scenario is named CPP model4, as shown in Fig. 16a. The area is decomposed into grids, and the A-type points, N-type points and NFZ are labeled. The decomposed and labeled area is shown in Fig. 16b.

Figure 17a-d illustrate the paths planned by BF,SP,WF and PSAACO on CPP model4, respectively. Figure 18a-d show the flight trajectories simulated with gazebo, corresponding to the paths planned by these four algorithms. Table 16 presents the simulation experiment results of these four algorithms. It can be seen from the figures and table:

- The flight trajectories of the UAV and the planned paths almost coincide, indicating that the path planned by the proposed algorithm is feasible for real flight.
- Compared with the other three algorithms, the PSAACO algorithm uses a shorter completion time, which proves that the PSAACO has better performance.
- The difference between the planned completion time and the simulated completion time is very small, ranging from 24 and 28.36 seconds, and the percentage of the difference is between 5.4% and 6.9%. It shows that the values of the main performance metrics used in this paper are almost consistent with the real values, and have high applicability.

Conclusion

In this paper, we propose the PSAACO algorithm to plan paths for areas of interest. The algorithm is improved by applying grid-mapping of the area, inversion and insertion operators, self-adaptive parameters, and parallel computing. Two metrics, completion time and energy consumption, are used to compare and evaluate the performance of PSAACO algorithm. The performance improvement experiments show that self-adaptive parameter setting and parallel computing can improve the performance of PSAACO algorithm, and the improvement is greater when the areas are large or the no-fly zones are complex. In the performance comparison experiments with other algorithms and existing works, PSAACO has excellent comprehensive performance on CPP, and has a great gain in the four indicators of total path length, total turning angle, energy consumption and completion time. The DFA is also proposed to deal with the NFZs in the areas. The DFA can dynamically add vertices and only calculate the changes caused by these vertices. However, the approach proposed in this paper also has some limitations, such as the use of parallel computing requires high-performance computing equipment; the algorithm runs slowly when the areas of interest are large. In the future works, we aim to speed up the algorithm, and evaluate the work when the areas are larger or the no-fly zones are more complex.

Abbreviations

UAV: Unmanned aerial vehicle; IoT: Internet of things; EC: Edge computing; CloudIoT(CoT): Cloud of things; CPP: Coverage path planning; NFZ: No-fly zone; PSAACO: Parallel self-adaptive ant colony optimization; DFA: Dynamical Floyd algorithm; BF: Back-and-forth; SP: Spiral; WF: Wavefront; EAS: Elitist ant system; ASRank: Rank based ant system; MMAS: MAX-MIN ant system; ACS: Ant colony system; TSP: Traveling salesman problem; JSP: Job-shop scheduling problem; VRP: Vehicle routing problem; QAP: Quadratic assignment problem.

Acknowledgements
The authors would like to thank to the reviewers for nice comments on the manuscript.

Authors' Contributions

Y.G. wrote the main manuscript text; K.G. and T.N. prepared figures and tables. All authors have read and agreed to the published version of the manuscript.

Authors' information

Not applicable.

Funding
Supported by National Key Research and Development Program of China (Grant No. 2018YFC1405700) and Industry University Research Cooperation Project of Jiangsu Province (Grant No. BY2019005).

Availability of data and materials

Some or all data, models, or codes that support the findings of this study are available from the corresponding author upon reasonable request.

Declarations

Ethics approval and consent to participate

Not applicable.

Consent for publication

Not applicable.

Competing interests

The authors declare that they have no competing interests.

Received: 27 May 2022 Accepted: 24 July 2022
Published online: 23 August 2022

References

1. Huang J, Zhang C, Zhang J (2020) A multi-queue approach of energy efficient task scheduling for sensor hubs. Chin J Electron 29(2):242–247. https://doi.org/10.1049/cje.2020.02.001
2. Chen Y, Xing H, Ma Z, et al (2022) Cost-efficient edge caching for noma-enabled iot services. China Commun
3. Xu X, Li H, Xu W et al (2022) Artificial intelligence for edge service optimization in internet of vehicles: A survey. Tsinghua Sci Technol 27(2):270–287. https://doi.org/10.26599/TST.2020.9010025
4. Huang J, Lv B, Wu Y et al (2022) Dynamic admission control and resource allocation for mobile edge computing enabled small cell network. IEEE Trans Veh Technol 71(2):1964–1973. https://doi.org/10.1109/TVT.2021.3133696
5. Qabil S, Waheed U, Awan SM, Mansoor Y, Khan MA (2019) A survey on emerging integration of cloud computing and internet of things. In: 2019 International Conference on Information Science and

Communication Technology (ICISCT). https://doi.org/10.1109/CISCT.2019.8777438
6. Aazam M, Khan I, Alsaffar AA, Huh E (2014) Cloud of things: Integrating internet of things and cloud computing and the issues involved. In: Proceedings of 2014 11th International Bhurban Conference on Applied Sciences Technology (IBCAST) Islamabad, Pakistan, 14th - 18th January, 2014, pp 414–419. https://doi.org/10.1109/IBCAST.2014.6778179
7. Chen Y, Zhao F, Lu Y, Chen X (2021) Dynamic task offloading for mobile edge computing with hybrid energy supply. Tsinghua Science and Technology 10, https://doi.org/10.26599/TST.2021.9010050
8. Chen Y, Gu W, Li K (2022) Dynamic task offloading for internet of things in mobile edge computing via deep reinforcement learning. Int J Commun Syst :e5154. https://doi.org/10.1002/dac.5154
9. Chen Y, Zhao F, Chen X, Wu Y (2022) Efficient multi-vehicle task offloading for mobile edge computing in 6g networks. IEEE Trans Veh Technol 71(5):4584–4595. https://doi.org/10.1109/TVT.2021.3133586
10. Chen Y, Liu Z, Zhang Y et al (2021) Deep reinforcement learning-based dynamic resource management for mobile edge computing in industrial internet of things. IEEE Trans Indust Inform 17(7):4925–4934
11. Xu X, Jiang Q, Zhang P, Cao X (2022) Game theory for distributed iov task offloading with fuzzy neural network in edge computing. IEEE Trans Fuzzy Syst. https://doi.org/10.1109/TFUZZ.2022.3158000
12. Li T, Li C, Lou C, Song L (2020) Wireless recommendations for internet of vehicles: Recent advances, challenges, and opportunities. Intell Converged Netw 1(1):1–17. https://doi.org/10.23919/ICN.2020.0005
13. Huang J, Tong Z, Feng Z (2022) Geographical poi recommendation for internet of things: A federated learning approach using matrix factorization. Int J Commun Syst :e5161https://doi.org/10.1002/dac.5161
14. Qi L, Lin W, Zhang X et al (2022) A correlation graph based approach for personalized and compatible web apis recommendation in mobile app development. IEEE Trans Knowl Data Eng. https://doi.org/10.1109/TKDE.2022.3168611
15. Sandhu AK (2022) Big data with cloud computing: Discussions and challenges. Big Data Min Analytics 5(1):32–40. https://doi.org/10.26599/BDMA.2021.9020016
16. Tong Z, Ye F, Yan M, Liu H, Basodi S (2021) A survey on algorithms for intelligent computing and smart city applications. Big Data Min Analytics 4(3):155–172. https://doi.org/10.26599/BDMA.2020.9020029
17. Catlett C, Beckman P, Ferrier N, Nusbaum H et al (2020) Measuring cities with software-defined sensors. J Soc Comput 1(1):14–17. https://doi.org/10.23919/JSC.2020.0003
18. Zhang W, Chen X, Jiang J (2021) A multi-objective optimization method of initial virtual machine fault-tolerant placement for star topological data centers of cloud systems. Tsinghua Sci Technol 26(1):95–111. https://doi.org/10.26599/TST.2019.9010044
19. Hou C, Wu J, Cao B, Fan J (2021) A deep-learning prediction model for imbalanced time series data forecasting. Big Data Min Analytics 4:266–278. https://doi.org/10.26599/BDMA.2021.9020011
20. Bouras MA, Farha F, Ning H (2020) Convergence of computing, communication, and caching in internet of things. Intell Converged Netw 1(1):18–36
21. Xu X, Tian H, Zhang X et al (2022) Discov: Distributed covid-19 detection on x-ray images with edge-cloud collaboration. IEEE Trans Serv Comput. https://doi.org/10.1109/TSC.2022.3142265
22. Wang Z, Tao J, Gao Y, Xu Y, Sun W, Li X (2021) A precision adjustable trajectory planning scheme for uav-based data collection in iots. Peer-to-Peer Netw Appl 14:655–671. https://doi.org/10.1007/s12083-020-01006-0
23. Xu J, Li D, Gu W et al (2022) Uav-assisted task offloading for iot in smart buildings and environment via deep reinforcement learning. Build Environ. https://doi.org/10.1016/j.buildenv.2022.109218
24. Ji X, Wang X, Niu Y, Shen L (2015) Cooperative search by multiple unmanned aerial vehicles in a nonconvex environment. Math Probl Eng 2015. https://doi.org/10.1155/2015/196730
25. Cho S, Park J, Park H, Kim S (2022) Multi-uav coverage path planning based on hexagonal grid decomposition in maritime search and rescue. Mathematics 10(1):83. https://doi.org/10.3390/math10010083
26. Cho SW, Park HJ, Lee H, Shim DH, Kim S (2021) Coverage path planning for multiple unmanned aerial vehicles in maritime search and rescue operations. Comput Ind Eng 161. https://doi.org/10.1016/j.cie.2021.107612
27. Cabreira TM, Franco CD, Ferreira PR, Buttazzo GC (2018) Energy-aware spiral coverage path planning for uav photogrammetric applications. IEEE Robot Autom Lett 3(4):3662–3668
28. Choi Y, Choi Y, Briceno S, Mavris DN (2020) Energy-constrained multi-uav coverage path planning for an aerial imagery mission using column generation. J Intell Robot Syst 97(1):125–139. https://doi.org/10.1007/s10846-019-01010-4
29. Almadhoun R, Taha T, Dias J, Seneviratne L, Zweiri Y (2019) Coverage path planning for complex structures inspection using unmanned aerial vehicle (uav). In: Springer, Cham
30. Biundini I, Pinto M, Melo A, Marcato AM, Honorio L (2022) Coverage path planning optimization based on point cloud for structural inspection. In: Khosravy M, Gupta NPN (eds) Frontiers in Nature-Inspired Industrial Optimization. Springer, Singapore, pp 141–156
31. Xiao S, Tan X, Wang J (2021) A simulated annealing algorithm and grid map-based uav coverage path planning method for 3d reconstruction. Electronics 10(7):853. https://doi.org/10.3390/electronics10070853
32. Shen Z, Song J, Mittal K, Gupta S (2022) CT-CPP: Coverage Path Planning for 3D Terrain Reconstruction Using Dynamic Coverage Trees. IEEE Robot Autom Lett 7(1):135–142. https://doi.org/10.1109/LRA.2021.3119870
33. Mokrane A, Braham AC, Cherki B (2019) Uav coverage path planning for supporting autonomous fruit counting systems. In: 2019 International Conference on Applied Automation and Industrial Diagnostics (ICAAID). IEEE, Elazig, p 1–5. https://doi.org/10.1109/ICAAID.2019.8934989
34. Tormagov T, Rapoport L (2021) Coverage path planning for 3d terrain with constraints on trajectory curvature based on second-order cone programming. In: Olenev NN, Evtushenko YG, Jaćimović M, Khachay M, Malkova V (eds) Advances in Optimization and Applications. OPTIMA 2021. Communications in Computer and Information Science, vol 1514. Springer, Cham, pp 258–272. https://doi.org/10.1007/978-3-030-92711-0_18
35. Arman N, Izbirak G, Vizvari B, Arkat J (2016) Complete coverage path planning for a multi-uav response system in post-earthquake assessment. Robotics 5(4):26. https://doi.org/10.3390/robotics5040026
36. Choset H (2001) Coverage for robotics - a survey of recent results. Ann Math Artif Intell 31:113–126
37. Wang K, Meng Z, Wang L, Wu Z, Wu Z (2019) Practical obstacle avoidance path planning for agriculture uavs. In: Wotawa F, Friedrich G, Pill I, Koitz-Hristov R, Ali M (eds) Advances and Trends in Artificial Intelligence. From Theory to Practice. IEA/AIE 2019. Lecture Notes in Computer Science, vol 11606. Springer, Cham, pp 196–203. https://doi.org/10.1007/978-3-030-22999-3_18
38. Nolan P, Paley DA, Kroeger K (2017) Multi-uas path planning for non-uniform data collection in precision agriculture. In: 2017 IEEE Aerospace Conference. IEEE, Big Sky, p 1–12. https://doi.org/10.1109/AERO.2017.7943794
39. Cabreira TM, Ferreira PR, Franco CD, Buttazzo GC (2019) Grid-based coverage path planning with minimum energy over irregular-shaped areas with uavs. In: International Conference on Unmanned Aircraft Systems (ICUAS). IEEE, Atlanta, p 758–767. https://doi.org/10.1109/ICUAS.2019.8797937
40. Ghaddar A, Merei A, Natalizio E (2020) Pps: Energy-aware grid-based coverage path planning for uavs using area partitioning in the presence of nfzs. Sensors 20(13):3742. https://doi.org/10.3390/s20133742
41. Chaari I, Koubâa A, Qureshi B et al (2018) On the robot path planning using cloud computing for large grid maps. 2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC). IEEE, Torres Vedras, p 225–230. https://doi.org/10.1109/ICARSC.2018.8374187
42. Cabreira TM, Brisolara LB, Ferreira PR (2019) Survey on coverage path planning with unmanned aerial vehicles. Drones 3(1):4. https://doi.org/10.3390/drones3010004
43. Galceran E, Carreras M (2013) A survey on coverage path planning for robotics. Robot Auton Syst 61(12):1258–1276
44. Barrientos A, Colorado J, Cerro JD, Martinez A, Rossi C, Sanz D, Valente J (2011) Aerial remote sensing in agriculture: A practical approach to area coverage and path planning for fleets of mini aerial robots. J Robot Syst 28:667–689
45. Valente J, Sanz D, Cerro JD, Barrientos A, Frutos M (2013) Near-optimal coverage trajectories for image mosaicing using a mini quad-rotor over irregular-shaped fields. Precis Agric 14:115–132

46. Ghaddar A, Merei A (2020) Eaoa: Energy-aware grid-based 3d-obstacle avoidance in coverage path planning for uavs. Futur Internet 12(2):29. https://doi.org/10.3390/fi12020029
47. Ghaddar A, Merei A (2019) Energy-aware grid based coverage path planning for uavs. In: SENSORCOMM 2019: The Thirteenth International Conference on Sensor Technologies and Applications. IARIA, Nice, p 34–45. https://www.thinkmind.org/index.php?view=article&articleid=sensorcomm_2019_2_30_10053
48. Colorni A, Dorigo M, Maniezzo V (1991) Distributed optimization by ant colonies. In: Proceedings of ECAL91 - European Conference on Artificial Life. ELSEVIER PUBLISHING, PARIS, p 134–142
49. Dorigo M, Stützle, T (2003) The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances. In: Glover F, Kochenberger GA (eds) Handbook of Metaheuristics. International Series in Operations Research & Management Science, vol 57. Springer, Boston. https://doi.org/10.1007/0-306-48056-5_9
50. Bullnheimer B, Hartl RF, Strauss C (1999) A new rank based version of the ant system - a computational study. CEJOR 7(1):25–38
51. Stutzle T, Hoos HH (2000) Max-min ant system. Futur Gener Comput Syst 16:889–914
52. Dorigo M, Gambardella LM (1997) Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans Evol Comput 1:53–66
53. Júnior O, Leal JE, Reimann M (2021) A multiple ant colony system with random variable neighborhood descent for the dynamic vehicle routing problem with time windows. Soft Comput 25:2935–2948. https://doi.org/10.1007/s00500-020-05350-4
54. Zhao B, Gao J, Chen K, Guo K (2018) Two-generation pareto ant colony algorithm for multi-objective job shop scheduling problem with alternative process plans and unrelated parallel machines. J Intell Manuf 29:93–108. https://doi.org/10.1007/s10845-015-1091-z
55. Shetty A, Shetty A, Puthusseri KS, Shankaramani R (2018) An improved ant colony optimization algorithm: Minion ant(mant) and its application on tsp. In: 2018 IEEE Symposium Series on Computational Intelligence (SSCI). IEEE, Bangalore, p 1219–1225. https://doi.org/10.1109/SSCI.2018.8628805
56. Maniezzo V, Colorni A (1999) The ant system applied to the quadratic assignment problem. IEEE Trans Knowl Data Eng 11(5):769–778
57. Gong Y, Wang W, Gong S (2022) A novel self-adaptive mixed-variable multiobjective ant colony optimization algorithm in mobile edge computing. Secur Commun Netw 2022. https://doi.org/10.1155/2022/4967775
58. Ilhan I, Gökmen G (2022) A list-based simulated annealing algorithm with crossover operator for the traveling salesman problem. Neural Comput & Applic 34:7627–7652
59. Pedemonte M, Nesmachnow S, Cancela H (2011) A survey on parallel ant colony optimization. Appl Soft Comput 11(8):5181–5197
60. Floyd RW (1962) Algorithm 97: Shortest path. Comm Acm 5(6). https://doi.org/10.1145/367766.368168
61. Warshall S (1962) A theorem on boolean matrices. J ACM 9(1):11–12
62. Nam LH, Huang L, Li XJ, Xu JF (2016) An approach for coverage path planning for uavs. In: 2016 IEEE 14th International Workshop on Advanced Motion Control (AMC). IEEE, Auckland, p 411–416. https://doi.org/10.1109/AMC.2016.7496385
63. Zelinsky A, Jarvis RA, Byrne JC, Yuta S (1993) Planning paths of complete coverage of an unstructured environment by a mobile robot. In: Proceedings of international conference on advanced robotics. p 533–538