An algorithm with exact bounds for coverage path planning in UAV-based search and rescue under windy conditions

Sina Kazemdehbashi[1] and Yanchao Liu[2*]

[1,2]Department of Industrial and Systems Engineering, Wayne State University, 4815 4th Street, Detroit, 48201, Michigan, USA.

*Corresponding author(s). E-mail(s): yanchaoliu@wayne.edu;

Abstract

Unmanned aerial vehicles (UAVs) are increasingly utilized in global search and rescue efforts, enhancing operational efficiency. In these missions, a coordinated swarm of UAVs is deployed to efficiently cover expansive areas by capturing and analyzing aerial imagery and footage. Rapid coverage is paramount in these scenarios, as swift discovery can mean the difference between life and death for those in peril. This paper focuses on planning the flight paths for multiple UAVs in windy conditions to efficiently cover rectangular search areas in minimal time. We address this challenge by dividing the search area into a grid network and formulating it as a mixed-integer program (MIP). We derive a precise lower bound for the objective function and develop a fast algorithm with a proven capability of finding either the optimal solution or a near-optimal solution with a constant absolute gap to optimality. Notably, as the problem complexity increases, our solution exhibits a diminishing relative optimality gap while maintaining negligible computational costs compared to the MIP approach. The fast execution speed of the algorithms is demonstrated by numerical experiments with area sizes up to 10000 grid cells.

Keywords: Coverage Path Planning, Unmanned Aerial Vehicle, Mixed-Integer Programming

1 Introduction

Thousands of people are reported missing and later found dead each year, due to being trapped or immobilized in harsh environments. For instance, approximately $4,000$ casualties in maritime environments have been reported each year since 2014 (Cho et al., 2021). To enhance the efficiency of life-saving operations, search and rescue (SAR) teams nowadays try to embrace cutting-edge technologies like UAVs and artificial intelligence to improve safety and operational efficiency (Martinez-Alpiste et al., 2021). Remote controlled or autopiloted UAVs are quicker to deploy, less expensive to operate and maintain and can reach more locations, compared to other means of search such as ones conducted by canine, on foot or by helicopter. UAVs are becoming the preferred equipment for many SAR missions (Lyu et al., 2023).

Despite the many benefits, using UAVs to conduct large area searches comes with several unique challenges, such as limitations and uncertainties brought about by the battery capacity, weather conditions, and air traffic safety considerations. The energy consumption, flight time and stability of small-sized multirotor UAVs are particularly susceptible to the effects of wind (Gianfelice et al., 2022). Furthermore, when multiple UAVs are employed in a SAR mission, the division of tasks and the planning of flight paths in the midst of the above constraints become nontrivial, and often require sophisticated treatment and calculation. Operations research that addresses the such challenges are termed *Coverage Path Planning (CPP)* and there is a rich literature about it (Bouzid et al., 2017; Di Franco and Buttazzo, 2016; Forsmo et al., 2013; Maza and Ollero, 2007). However, most of the studies ignored weather conditions. We argue that minimizing the overall time to search an area exhaustively is a primary objective in SAR type of operations, and the relation between flight and wind directions plays an important role in optimizing that objective. This argument is corroborated in Coombes et al. (2018), which presented a wind-aware area survey method to optimally cover an area with a single UAV.

In this paper, we aim to fill a research gap by solving a practical version of the CPP problem that involves using multiple UAVs to cover a rectangular area with explicit consideration for wind condition. Specifically, we propose to discretize the search area into a grid of square cells whereas the dimension of a cell matches the flight altitude and camera aperture configurations of the mission. The orientation of this artificial grid is set in a way such that UAVs traversing the grid in a Von Neumann fashion (to be explained in Sect. 3.1) will fly in a direction either parallel to (same or opposite) or perpendicular to the wind direction. On top of this grid, we formulate the multiple-UAV CPP problem as a mixed-integer programming model which can be solved using commercial solvers such as CPLEX and GUROBI. To significantly shorten the computing time for larger instances, we develop a specialized algorithm that can guarantee a feasible solution with a provable performance bound. More specifically, the solution produced by our proposed algorithm either yields the minimal coverage time or yields a coverage time which is exactly T_p longer than the minimum, where T_p is the time it takes for a UAV to traverse one cell in the direction perpendicular to the wind.

The organization of the paper is as follows: Section 2 reviews the related literature. Section 3 defines the problem and presents a MIP formulation of the problem. In Sect. 4, we derive a mathematical formula to obtain the lower bound of the problem's objective, and introduce an efficient algorithm to construct a feasible solution whose objective is either the same as or close to the lower bound, thereby solving the problem much faster than a commercial MIP solver. Section 5 proves that the proposed method can be extended to cases utilizing the Moore neighborhood connectivity (to be defined in Sect. 3.1). Validation experiments and sensitivity analysis are presented in Sect. 6. Section 7 concludes the paper and suggests several extensions and new developments for future work. Some proofs, auxiliary procedures and additional experiments are put in the Appendices.

2 Literature Review

In recent years, UAVs are utilized in a growing variety of tasks, including delivering restaurant food (Liu, 2019), providing emergency network coverage for disaster inflicted areas (Park et al., 2024), performing maritime search and rescue (Cho et al., 2021), and aiding in humanitarian logistics (Zhang and Li, 2023). In search and rescue operations, the UAV mission planning challenge can often be modeled as a CPP problem, the primary objective of which is to survey an entire target region in minimal time. Cabreira, Brisolara, and Ferreira (2019) reviewed studies on CPP using UAVs with several classification criteria, including the shape of the search area, how the search area is decomposed into smaller, more manageable parts, the performance metrics being used to guide the path planning, and how the search patterns are formed. Among these criteria, the area decomposition method is the primary factor that determines the nature of the mathematical model and solution approach for the problem. Therefore, we categorize CPP methods into two types based on this factor: ones that employed an exact cellular decomposition of the search area, and ones that employed an approximate (grid-based) decomposition of the search area.

In the exact cell decomposition method, the area of interest is divided into several sub-areas using linear boundaries, and the sub-areas are called cells. Latombe (1991) presented a trapezoidal decomposition that divided the target region into trapezoidal cells, whereas each cell could be covered with simple back-and-forth motions. Choset (2000) proposed an improved method called the boustrophedon decomposition that attempted to merge the trapezoidal cells into larger non-convex cells to create opportunities for path length reduction when the robots engaged in back-and-forth motions to cover the cells. Kong et al. (2006) utilized the boustrophedon cell decomposition to address the multi-robot coverage problem. Jiao et al. (2010) presented a method for the CPP problem in polygonal areas. They defined the width of a convex polygonal shape and demonstrated that a UAV should fly along the vertical direction of the width to achieve a coverage path with the least number of turns. Then, they proposed a convex decomposition algorithm to partition a concave area into convex subregions by minimizing the sum of the widths of those subregions. They developed a subregion connection algorithm to determine which combination of subregions has the minimum traversal path to cover the entire area. Li et al. (2011) proposed a method to address the coverage path planning for UAVs in a polygonal area. They showed that, in terms of duration, energy, and path length, turning is an inefficient motion and should be avoided. The authors presented a path with minimal turns for a UAV. They developed a decomposition algorithm to convert a concave area into convex subregions and proposed a subregion connection algorithm to connect adjacent subregions. Di Franco and Buttazzo (2015) proposed an energy-aware path planning algorithm to fully cover a given area while considering other constraints, namely the available energy, the minimum spatial resolution for the pictures, and the maximum camera sampling period. Furthermore, the same authors proposed an energy model for a single UAV based on real measurements, and used this model to reduce energy consumption in path planning, see Di Franco and Buttazzo (2016). This method assumes back-and-forth motions and determines the UAV's speed to minimize energy consumption, and then the estimated energy consumption is checked for sufficiency to sustain the generated path. Coombes et al. (2018) proposed a new method for planning coverage paths for fixed-wing UAV aerial surveys. They considered windy conditions and proved that flying perpendicular to the wind direction confers a flight time advantage over flying parallel to the wind direction. Additionally, they used dynamic programming to find time-optimal convex decomposition within a polygon. Bähnemann et al. (2021) extended the boustrophedon coverage planning by considering different sweep directions in each cell to identify the efficient sweep path. Additionally, they utilized the Equality Generalized Traveling Salesman Problem (E-GTSP) to formulate and find the minimum total path in the adjacency graph.

In addressing the CPP problem, ensuring the complete coverage of the given area is one of the main concerns. This is commonly accomplished by employing cellular decomposition in the area of interest, where the target area is divided into cells to make coverage simpler (Choset, 2001). In addition to exact cell decomposition, approximate cellular decomposition, or grid-based decomposition, is a common approach used in the literature, where the area of interest is divided into a set of cells, all of which have the same size and shape (Choset, 2001). In the grid-based decomposition scheme, some borderline regions outside the area of interest may be included in grid cells, leading to a waste of time and resources which is proportional to the granularity of the grid. Despite the loss of accuracy due to discretization, the grid-based decomposition scheme has been adopted in many research works. In our opinion, the advantage of grid-based approach includes the ease of managing trajectory conflicts among multiple UAVs, the availability of well-developed frameworks such as mixed-integer programming to analyze the algorithmic performances, and its flexibility to accommodate exact and heuristic enhancements for the path planning task. This viewpoint is corroborated by the appreciable volume of studies centered on grid-based decomposition.

Barrientos et al. (2011) proposed an integrated tool using a fleet of unmanned aircraft capable of capturing georeferenced images. They considered both regular and irregular grid shapes and utilized the Wavefront algorithm in their approach. Their tool included several components such as task partitioning and allocation, the CPP algorithm, and robust flight control. Valente et al. (2013) proposed a path planning tool that converts an irregular area into a grid graph and generates near-optimal trajectories for UAVs by minimizing the number of turns in their paths. Nam et al. (2016) used grid-based decomposition and wavefront algorithm for a single UAV and considered not only the number of turns but also the route length. Balampanis et al. (2017) proposed a method to decompose and partition a complex coastal area and generate a list of waypoints for multiple heterogeneous unmanned aircraft systems (UAS) to cover the area. In their method, the search area was partitioned into several subregions, with each subregion assigned to a UAS for coverage. Furthermore, each subregion was decomposed into a grid of triangular cells, for the designated UAS to follow a moving pattern that connects the borders of the sub-area to the inner regions. Bouzid et al. (2017) converted a quadrotor optimal coverage planning problem in areas with obstacles into a Traveling Salesman Problem (TSP) that minimizes the overall energy consumption, and used Genetic Algorithms (GA) to solve it. Cabreira, Ferreira, et al. (2019) proposed an energy-aware grid-based covering path planning algorithm (EG-CPP) to minimize the energy consumption of UAVs for covering irregular-shaped areas. They stated that solely considering turns would inadequately approximate the energy consumption of UAV paths. Consequently, they improved the cost function proposed in Valente et al. (2013), which previously only accounted for the number of turns. This enhancement resulted in a 17% energy saving in real flight tests. Cho et al. (2021) considered heterogeneous UAVs in polygon-shaped areas in maritime search and rescue. They proposed a grid-based area decomposition method to convert an area into a graph, and formulated a MIP model on the graph to find the coverage path with minimal completion time. Moreover, they introduced a randomized search heuristic (RSH) algorithm to shorten the computation time for large-scale instances while preserving a small optimality gap. In their subsequent work Cho et al. (2022), the authors compared the use of hexagonal cells and square cells in a MIP framework, and concluded that the former is more effective for generating paths to minimize the coverage time. The exponential growth in computational complexity with the size of the search area was observed in their experiments. In this paper, we also use a MIP model as comparison baseline, but develop a fast algorithm with exact performance bounds instead of a heuristic one for our main contribution. Ai et al. (2021) used reinforcement learning in coverage path planning for a vessel agent, in which wind and water flow data were collected to make a probability map of the true location of the target in the grid. The authors showed that their approach outperforms some of the previous approaches in three simulated scenarios. Song et al. (2022) presented a method for UAV path planning, focusing on visiting important points within the search area instead of exhaustive full coverage. They decomposed the area into a grid and assign a score to each cell by using a classifier, generating a new map called heatmap. They tried to enhance the efficiency of rescue operations by incorporating this heatmap into path planning because of the limited endurance of UAVs and the constrained rescue time following a disaster. They proposed three path planning algorithms, conducting comparative analyses among them and an existing algorithm. One of the proposed algorithms demonstrated superior performance; however, none of the algorithms could ensure optimality. Ahmed et al. (2023) developed two methods, a greedy algorithm and Simulated Annealing (SA), to address the CPP problem for multiple

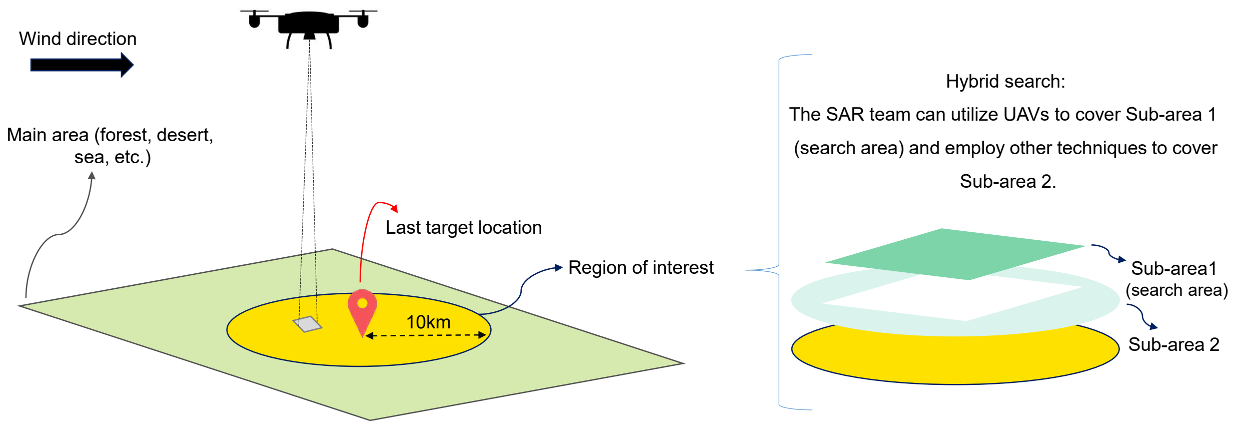
Wind direction
Main area (forest, desert, sea, etc.)
Last target location
10km
Region of interest
Hybrid search:
The SAR team can utilize UAVs to cover Sub-area 1 (search area) and employ other techniques to cover Sub-area 2.
Sub-area1 (search area)
Sub-area 2

Fig. 1: Suppose a SAR team receives a report of a missing mid-age hiker whose last contact was about two hours ago. Assuming an average human walking speed of 5 km/h, the radius of the region of interest would be approximately 10 km.

UAVs. They formulated the problem as a MIP model to minimize energy consumption. They showed for small-scale cases CPLEX had better performance, but in large-scale problems, SA outperformed others to minimize the overall energy consumption.

The contributions of our study are summarized as follows. First, for the multi-UAV CPP problem we present a mathematical formula to calculate the lower bound of the coverage time in windy conditions. Second, we propose a constructive algorithm that can rapidly generate near-optimal concurrent coverage paths for large rectangular grids, and prove the solution's near optimality via an exhaustive analysis of data scenarios that may be encountered in the algorithm's execution. Third, we conduct thorough numerical experiments of varying scales to demonstrate the computational advantage and broad usability of the proposed algorithm in addressing complex search and rescue problems utilizing multiple UAVs.

3 Problem Formulation

Consider a typical scenario of searching for a missing person in rural, mountainous and otherwise scantly inhabited areas: The SAR team received a call for help with locating a mid-age hiker whose last approximate location was recorded by a cellular tower two hours ago. By assuming an average walking speed of 5 km/h in this terrain, the team postulated a circular area of 10 km radius that must be searched as quickly as possible. To expedite the search, mixed resources would be utilized, including a fleet of UAVs to cover the central (presumably hard-to-reach) region and canine and on-foot squads to sweep in from the peripheral areas simultaneously. The division of tasks is demonstrated in Fig. 1. An alternative search mode might be searching the entire region by UAVs, in which case a square (or rectangular) area that closely approximates or tightly encloses the region could be set as the target area for aerial search. In either context, we are concerned with planning the flight paths for the UAVs in a uniform wind field, with the objective of covering (i.e., having a clear aerial image of) all locations in a given rectangular area in minimal time. The UAVs are assumed identical and hence fly at the same airspeed. Collision avoidance among these UAVs must be explicitly considered.

3.1 Grid-based decomposition

To plan the flight paths of multiple UAVs in the rectangular shaped search area, we first decompose the search area into equal-sized square tiles (or cells) so that each tile fits in the image scope of the UAV's downward facing camera. The actual side length of the tile, together with the flight altitude of the UAV, is determined by factors such as the field-of-view (FoV) angle of the camera, the camera's resolution and the required representation resolution (centimeters of the ground surface per pixel), as demonstrated in Fig. 2a. Pertinent to our modeling framework, each cell is denoted by its central coordinates; for example, cell (i, j) signifies that its central coordinates are located at (i, j), as depicted in Fig. 2b.

In addition, the following assumptions about the search area is made:

Assumption 1: The $n \times m$ search area represents a rectangular region consisting of n cells along its length (X-axis) and m cells along its width (Y-axis), as depicted in Fig. 2b, and m is greater than or equal to the number of UAVs, denoted by q.

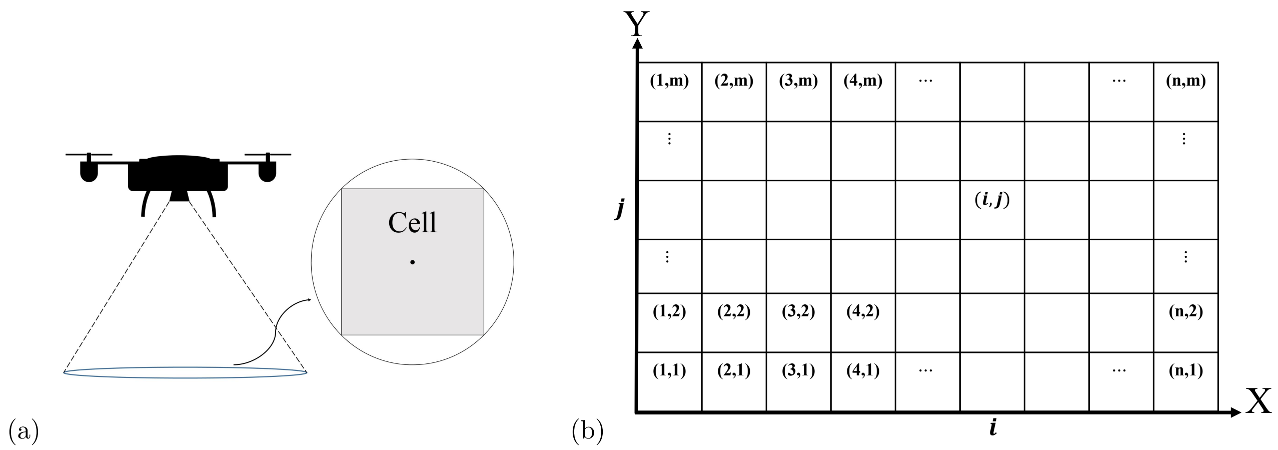

Fig. 2: (a) The camera's field of view, aligned with a cell in the grid; (b) Cell indexing convention for an $(n \times m)$ rectangular search area.

Assumption 2: The wind speed is constant over the search area and during the entire search time, and the wind direction is parallel to the X-axis of the search area, flowing from west to east, as shown in Fig. 3. Since the search area's orientation can be arbitrary, the latter assumption causes no loss of generality.

In the designated search area, two connectivity types are considered. First, the Von Neumann neighborhood enables a UAV to visit adjacent cells labeled as $1, 3, 5$, and 7 (as shown in Fig. 3). Second, the Moore neighborhood allows the UAV to visit all eight neighboring cells. Throughout the subsequent sections, our analysis and proposed solution are based on the utilization of the Von Neumann neighborhood. In Sect. 5, we demonstrate the validity of our proposed solution even when employing the Moore neighborhood as the selected connectivity type.

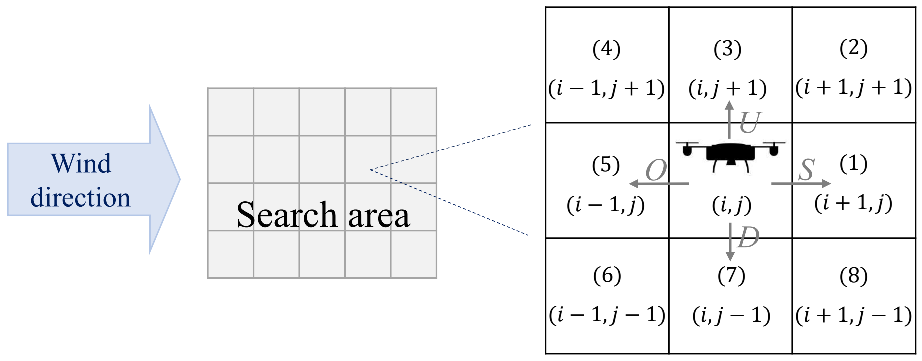

Fig. 3: Illustration of cell labeling for neighbors under Assumption 2.

Before continuing to the mathematical formulation, let us define some terms used frequently in the rest of the paper. The term "S move" means a UAV goes from one cell to its neighboring cell in the wind's direction (cell labeled 1 in Fig. 3). Conversely, "O move" refers to a UAV moving from a cell to its neighbor opposite to the wind (cell labeled 5 in Fig. 3). "U move" denotes a UAV moving upward to its upper neighbor (cell labeled 3 in Fig. 3), and "D move" represents the same action but in the downward direction (cell labeled 7 in Fig. 3). Additionally, the term "P move" signifies UAV transitions to neighboring cells perpendicular to the wind's direction (cells labeled 3 or 7 in Fig. 3); in other words, it's either a U move or a D move. The term *Mission Time* denotes the duration required for a UAV to visit its assigned cells for coverage. Also, the term *Operation Time* represents the duration required for a fleet of UAVs to fully cover a search area. Table 1 defines the parameters that are used consistently throughout the paper.

3.2 Mathematical formulation - a baseline

We first formulate the problem as a mixed-integer programming (MIP) model utilizing the notations provided in Table 2. This model helps us to characterize the data, decisions and their relations more rigorously, and

Table 1: Parameters that define a problem instance

Notations	
n	Number of cells along the length (X-axis) of the search area
m	Number of cells along the width (Y-axis) of the search area
q	Number of UAVs
T_s	Time required for a UAV to perform an S move
T_p	Time required for a UAV to perform a P move
T_o	Time required for a UAV to perform an O move

will serve as a baseline for validating computational results.

Table 2: MIP model notations

Sets	
I	Set of X-coordinates for cells
J	Set of Y-coordinates for cells
K	Set of UAVs
S	Set of indices $\{1, 2, \ldots, n \times m\}$ for movement steps
Variables	For $i \in I, j \in J, k \in K,$ and $s \in S$ where applicable
x_{kijs}	1 if UAV k visits cell (i, j) in step s, otherwise 0
y_{ks}	1 if UAV k is located outside the area in step s, otherwise 0
p_{ks}	1 if UAV k moves perpendicular to the wind direction from step $s-1$ to s, otherwise 0
r_{ks}	1 if UAV k moves in the wind direction from step $s-1$ to s, otherwise 0
l_{ks}	1 if UAV k moves against the wind direction from step $s-1$ to s, otherwise 0
t_{opr}	Operation time, total time to cover the whole area

$$Min \quad t_{\text{opr}} \tag{1}$$

$$s.t. \quad \sum_{i \in I} \sum_{j \in J} x_{kijs} + y_{ks} = 1 \quad \forall k \in K,\ \forall s \in S \tag{2}$$

$$\sum_{k \in K} \sum_{s \in S} x_{kijs} \geq 1 \quad \forall i \in I,\ \forall j \in J \tag{3}$$

$$\sum_{k \in K} x_{kijs} \leq 1 \quad \forall i \in I,\ \forall j \in J,\ \forall s \in S \tag{4}$$

$$x_{kijs} - \left(x_{k(i-1)j(s-1)} + x_{k(i+1)j(s-1)} + x_{ki(j-1)(s-1)} + x_{ki(j+1)(s-1)}\right) \leq 0$$
$$\forall i \in I \setminus \{1, |I|\},\ \forall j \in J \setminus \{1, |J|\},\ \forall k \in K,\ \forall s \in S \setminus \{1\} \tag{5}$$

$$x_{kijs} - \left(y_{k(s+1)} + x_{k(i-1)j(s+1)} + x_{k(i+1)j(s+1)} + x_{ki(j-1)(s+1)} +\right.$$
$$\left. x_{ki(j+1)(s+1)}\right) \leq 0 \quad \forall i \in I \setminus \{1, |I|\},\ \forall j \in J \setminus \{1, |J|\},\ \forall k \in K,\ \forall s \in S \setminus \{|S|\} \tag{6}$$

$$\sum_{j \in J} x_{kij(s-1)} + \sum_{j \in J} x_{k(i+1)js} - 1 \leq r_{ks} \quad \forall i \in I \setminus \{|I|\},\ \forall s \in S \setminus \{1\},\ \forall k \in K \tag{7}$$

$$\sum_{j \in J} x_{kij(s-1)} + \sum_{j \in J} x_{k(i-1)js} - 1 \leq l_{ks} \quad \forall i \in I \setminus \{1\},\ \forall s \in S \setminus \{1\},\ \forall k \in K \tag{8}$$

$$\sum_{i \in I} x_{kij(s-1)} + \sum_{i \in I} x_{ki(j+1)s} - 1 \leq p_{ks} \quad \forall j \in J \setminus \{|J|\},\ \forall s \in S \setminus \{1\},\ \forall k \in K \tag{9}$$

$$\sum_{i \in I} x_{kij(s-1)} + \sum_{i \in I} x_{ki(j-1)s} - 1 \leq p_{ks} \quad \forall j \in J \setminus \{1\},\ \forall s \in S \setminus \{1\},\ \forall k \in K \tag{10}$$

$$p_{ks} + r_{ks} + l_{ks} + y_{ks} = 1 \quad \forall k \in K,\ \forall s \in S \setminus \{1\} \tag{11}$$

$$t_{\text{opr}} \geq T_p \cdot \sum_{s \in S \setminus \{1\}} p_{ks} + T_s \cdot \sum_{s \in S \setminus \{1\}} r_{ks} + T_o \cdot \sum_{s \in S \setminus \{1\}} l_{ks} \quad \forall k \in K \tag{12}$$

The objective (1) minimizes the operation time, and Constraint (2) ensures that each UAV is either positioned within a cell or outside of the search area. Constraint (3) and (4) ensures that all cells are visited at least once and at any time step at most one UAV can be in the same cell (collision avoidance). Constraint (5) states that a UAV can enter a cell only from its neighboring cells, and Constraint (6) indicates that a UAV can only move to adjacent cells or leave the search area. Constraints (7) and (8) indicate moves aligned with and against the wind direction, respectively; also, Constraints (9) as well as (10) specify moves perpendicular to the wind direction. According to Constraint (11), a UAV within the search area cannot simultaneously execute two or more types of moves. Finally, Constraint (12) states that the operation time must be greater than or equal to the maximum of UAVs' mission time, which is calculated on the right-hand side of the inequality.

Note that the starting positions of UAVs are not specified in the above model formulation. In an axis-aligned wind field, minimizing the objective function will automatically place the starting points along the upwind boundary of the search area. In the same vein, we do not need explicit constraints to keep flight paths inside the search area, as flying outside the search area amounts to a waste of time and will be automatically penalized via the objective function.

4 Main Method

In general, MIP problems are intractable and the convergence of branch-and-bound type of algorithms takes exponentially longer as the instance size grows. The above MIP model is no exception - commercial solvers running the above MIP model can only solve small-scale instances (see Section 6.1).

We describe an ingenious approach to significantly expedite the search for the optimal solution. While our algorithm cannot guarantee to always converge to the optimal solution, we can, however, accurately quantify the gap between the found solution and the optimal solution. In fact, the optimality gap resulted from our algorithm will be shown to always take one of two values, zero and T_p. Therefore, as the search grid grows in size (and hence the optimal value which represents the total time it takes to cover all cells), the relative gap will diminish to zero.

Notations pertinent to the following discussion are defined in Table 3. Most constraints in the MIP formulation, i.e., constraints (5) to (11), are for defining a sequence of ordered cells that constitutes a path. We first simplify the notation by introducing a decision variable p_k which represents a path of length k, where the definition of a path is absorbed in the definition of the variable, see Table 3. We further drop the collision avoidance constraint (4), to arrive at a relaxed problem with the aim of covering all grid cells in a $n \times m$ search area using q paths (overlapping allowed). We term this relaxed problem as Multiple UAV Coverage Path Planning (MUCPP).

Table 3: MUCPP model and propositions notations

Symbol	Meaning
$\bar{C}$	Set of cells in the $n \times m$ search area, $\{(1,1),(1,2),...,(n,m)\}$
N_c	Set of neighboring cells for cell c where $c \in \bar{C}$, $\lvert N_c\rvert \leq 4$
p_k	Sequence of k ordered cells, $(c_{(1)}, c_{(2)}, ..., c_{(k)})$, in which $c_{(i)} \in \bar{C}$ for $i = 1,2,3,...,k$, $c_{(i+1)} \in N_{c_{(i)}}$ for $i = 1,2,...,k-1$, called *path of length k*
p'_k	Set of cells within path p_k
P_k	Set of all paths of length k
P	Set of all paths, $\cup_k P_k$
$t_{c,c'}$	Time taken to move from cell c to cell c', where $c \in \bar{C}$ and $c' \in N_c$ ($t_{c,c'} \in \{T_s, T_o, T_p\}$)
t_{p_k}	Time taken to cover cells in path p_k by a UAV, equal to $\sum_{i=1}^{k-1} t_{c_{(i)},c_{(i+1)}}$ for $p_k \in P_k$
d_i	Number of cells assigned to UAV i, where $i \in \{1,2,\ldots,q\}$
$\vec{v}_a$	2-D vector representing the airspeed of the UAV
$\vec{v}_w$	2-D vector representing the wind speed
$\vec{v}_g$	2-D vector representing the ground speed of the UAV Note that $\vec{v}_g = \vec{v}_a + \vec{v}_w$.

MUCPP :

$$\min_{p_{d_1}, p_{d_2}, \ldots, p_{d_q}} \quad \max_{i \in \{1,2,\ldots,q\}} t_{p_{d_i}} \tag{13}$$

$$s.t. \quad \cup_{i=1}^{q} p'_{d_i} = \bar{C} \tag{14}$$

$$p_{d_i} \in P \qquad \forall i \in \{1,2,\dots,q\} \tag{15}$$

The objective (13) is to minimize the maximum time taken by UAVs to cover cells in their assigned path. Constraint (14) requires that the solution must cover all cells, and Constraint (15), along with variable definitions in Table 3, ensures the sequential connectivity of cells that constitute a valid path. To derive a lower bound, denoted as "LB", for the MUCPP problem we replace Constraint (14) by a weaker one to create a relaxed version of the problem called R-MUCPP (Relaxed Multiple UAVs Coverage Path Planning).

$$\begin{aligned}
&\text{R-MUCPP}: \\
&\min_{p_{d_1}, p_{d_2}, \dots, p_{d_q}} && \max_{i \in \{1,2,\dots,q\}} t_{p_{d_i}} && && (16)\\
&s.t. && \sum_{i=1}^{q} d_i \geq nm && d_i \in \mathbb{N} && (17)\\
& && p_{d_i} \in P && \forall i \in \{1,2,\dots,q\} && (18)
\end{aligned}$$

In the R-MUCPP problem, the objective (16) and Constraint (18) are inherited from the original MUCPP formulation. Constraint (17) requires that the total number of cells assigned to UAVs must be greater than or equal to the number of cells within the search area. If a solution satisfies Constraint (14), it is evident that it will also satisfy Constraint (17). In the next step, Section 4.1 presents two propositions that form the basis for obtaining the solution to the R-MUCPP problem. The optimal objective value of R-MUCPP will serve as the LB for MUCPP, thus a valid LB for the MIP.

4.1 Lower bound derivation

Let $\bar{D}$ represent the distance between the centers of two adjacent cells, and assume $\|\vec{v}_a\| = v_a$, $\|\vec{v}_w\| = v_w$, and $v_a > v_w \geq 0$. Then, the values of T_s, T_p, and T_o can be calculated using Eq. (19).

$$T_s = \frac{\bar{D}}{v_a + v_w} \qquad T_p = \frac{\bar{D}}{\sqrt{v_a^2 - v_w^2}} \qquad T_o = \frac{\bar{D}}{v_a - v_w} \tag{19}$$

Furthermore, we have $T_s \leq T_p \leq T_o$ because of $v_a + v_w \geq \sqrt{v_a^2 - v_w^2} \geq v_a - v_w$. To introduce and prove the LB for the MUCPP problem, we need to establish some preliminary results.

Lemma 1. *Let A, B, K, H, and N be non-negative real numbers satisfying $A \leq B \leq K$, $A + K \geq 2B$, and $H \geq N$. Then, the optimal value for (x_1, x_2, x_3) in the following linear programming problem is $(x_1^*, x_2^*, x_3^*) = (N, H - N, 0)$.*

$$\begin{aligned}
&\min && Ax_1 + Bx_2 + Kx_3 && && (20)\\
&s.t. && x_1 - x_3 \leq N && && (21)\\
& && x_1 + x_2 + x_3 = H && && (22)\\
& && x_i \geq 0 && i \in \{1,2,3\} && (23)
\end{aligned}$$

Proof.

$$\begin{aligned}
&\min_{x_1,x_2,x_3 \geq 0} Ax_1 + Bx_2 + Kx_3 \\
=&\min_{x_1,x_3 \geq 0} Ax_1 + B(H - x_1 - x_3) + Kx_3 \\
=&\min_{x_1,x_3 \geq 0} (A - B)x_1 + (K - B)x_3 + BH \\
\geq&\min_{x_3 \geq 0} (A - B)(N + x_3) + (K - B)x_3 + BH \\
=&\min_{x_3 \geq 0} (A + K - 2B)x_3 + (A - B)N + BH \\
=&AN + (H - N)B
\end{aligned}$$

where the first equality is by applying (22), the second equality is by collecting terms, the inequality is by (21) and the assumption that $(A - B) \leq 0$, and the last step is because of $A + K - 2B \geq 0$ and the nonnegativity of x_3. The lower bound of the objective value is attained when $x_3 = 0$ and the third line holds at equality, i.e., when x_1 is equal to $x_3 + N$. The solution $(x_1, x_2, x_3) = (N, H - N, 0)$ is feasible and makes the objective function achieves a valid lower bound, therefore, it is optimal. □

Proposition 1. *The time taken to fly a path of length d is at least T, defined as*

$$T = \begin{cases} (d-1)T_s & \text{if } d \leq n \\ (n-1)T_s + (d-n)T_p & \text{if } d > n \end{cases}$$

Proof. We know that

$$\begin{aligned} p_d &= (c_{(1)}, c_{(2)}, \ldots, c_{(d)}) \\ t_{p_d} &= \sum_{i=1}^{d-1} t_{c_{(i)}, c_{(i+1)}} \\ t_{c_{(i)}, c_{(i+1)}} &\in \{T_s, T_p, T_o\} \end{aligned}$$

Consider the first state $(d \leq n)$, since $t_{c_{(i)}, c_{(i+1)}}$ can take three values, the minimum of the t_{p_d} would be attained if we choose T_s for $t_{c_{(i)}, c_{(i+1)}}$, for all $i \in \{1, \ldots, d-1\}$. Hence, in this state, the mission time, t_{p_d}, cannot be less than $(d-1)T_s$.

In the second case, for a clearer explanation, the formula to calculate t_{p_d} can be written as $t_{p_d} = T_s x_1 + T_p x_2 + T_o x_3$, where x_1, x_2, and x_3 are defined as the number of S moves, P moves, and O moves, respectively, with the constraint $x_1 + x_2 + x_3 = d - 1$.

By the definitions given in Eq. (19), we have

$$T_s + T_o = \frac{2v_a \bar{D}}{v_a^2 - v_w^2} = \frac{v_a}{\sqrt{v_a^2 - v_w^2}} \frac{2\bar{D}}{\sqrt{v_a^2 - v_w^2}} \geq \frac{2\bar{D}}{\sqrt{v_a^2 - v_w^2}} = 2T_p$$

Furthermore, we have $T_s \leq T_p \leq T_o$ by assumption. Therefore, Lemma 1 can be applied; for this purpose, as it is mentioned before, consider x_1 as the number of S moves, x_2 as the number of P moves, x_3 as the number of O moves, T_s as A or x_1 coefficient, T_p as B or x_2 coefficient, and T_o as K or x_3 coefficient. In addition, it is supposed that the length of the area has n cells, which means the summation of the S moves and O moves should always be less than $n - 1$ to guarantee that the UAV will stay in the search area, it can be stated $x_1 - x_3 \leq n - 1$. It is clear that $x_1 + x_2 + x_3 = d - 1$ due to the length of the path, and additionally, it is considered $d - 1$ as H and $n - 1$ as N in Lemma 1. Based on the Lemma 1 solution, the optimal values are $x_1^* = n - 1, x_2^* = d - n, x_3^* = 0$; therefore, the minimum possible time for a path of length d is $(n-1)T_s + (d-n)T_p$. □

Lemma 2. *For any positive integer N, if we express it as the sum of d non-negative integers, then the largest summand in the summation cannot be smaller than $\lceil \frac{N}{d} \rceil$.*

Proof. Assume for contradiction that the largest component is equal to $Q := \lceil \frac{N}{d} \rceil - k, (k \geq 1)$. There is a $\alpha \in [0, 1)$ such that $\lceil \frac{N}{d} \rceil = \frac{N}{d} + \alpha$, so Q can be written as $Q = \frac{N}{d} + \alpha - k$. All of the d summands are less than or equal to Q, so their sum cannot exceed $dQ = N + d(\alpha - k)$, a quantity strictly less than N. This is a contradiction. □

Now, we can state a key proposition as follows.

Proposition 2. *Let n, m, q be positive integers satisfying $q \leq m$, then the time that is required to cover all cells of the $n \times m$ search area by q UAVs cannot be less than $(n-1)T_s + (\lceil \frac{nm}{q} \rceil - n)T_p$.*

Proof. It is clear that the time required to fully cover the $n \times m$ search area by q UAVs, i.e., operation time, must be greater than or equal to the maximum mission time of the UAVs. Based on Lemma 2, we establish that the maximum number of cells assigned to UAVs must be no less than $\lceil \frac{nm}{q} \rceil$. Consequently, in accordance with Proposition 1, the maximum mission time of the UAVs cannot be less than $(n-1)T_s + (\lceil \frac{nm}{q} \rceil - n)T_p$. □

Table 4: Notations used in the NOPP algorithm

Symbol	Meaning
F	Set of uncovered cells
C	Arbitrary cell in the search area, its coordinates (coordinates of the center) are indicated by (c_x, c_y)
NP	Number of available (remaining) P moves for a UAV
NU	Number of U moves made by a UAV
ND	Number of D moves made by a UAV
A	Number of cells assigned to a UAV to cover
SP	Cell from which a UAV starts its path, referred as *Starting point*
$\bar{O}$	Binary variable denotes the parity of the initial value of NP: True for odd, False for even
c_x	X-coordinate of cell C
c_y	Y-coordinate of cell C
path	Sequence of cells' coordination traversed and covered by a UAV
*path**	Sequence of moves executed by a UAV, elements must be "U", "D", or "S" such that: • "U" means U move • "D" means D move • "S" means S move
$\cup^*$	Operator between two sequences with the following definition: let S_1 and S_2 be two sequences such that $S_1 = (a_1, a_2, \ldots, a_z)$ and $S_2 = (b_1, b_2, \ldots, b_w)$, for $z, w \in \mathbb{N}$; then, $S_1 \cup^* S_2 = (a_1, a_2, \ldots, a_z, b_1, b_2, \ldots, b_w)$

Note: For further clarification regarding *path* and *path**, let's consider a scenario in which a UAV starts from the bottom-left corner of Fig. 2b at cell $(1,1)$ and covers the subsequent four cells above it. In this case, *path* is defined as $((1,1),(1,2),(1,3),(1,4),(1,5))$, while *path** corresponds to the UAV's four U moves, represented as ("U", "U", "U", "U").

According to Proposition 2, the optimal solution of the R-MUCPP problem, acting as the lower bound of the MUCPP problem (LB), can be derived using the following formula: $(n-1)T_s + (\lceil \frac{nm}{q} \rceil - n)T_p$ (referred to as the "LB formula" throughout the rest of this paper). In Sect. 4.2, we will employ the LB formula as a key insight in the development of the solution algorithm for the MUCPP problem.

4.2 Near-optimal path planning algorithm (NOPP)

We develop a set of procedures that deterministically construct feasible solutions of an objective value lying in the set $\{\text{LB}, \text{LB} + T_p\}$, where LB represents the lower bound established for the MUCPP problem (explained in Sect. 4.1). We call the procedures near-optimal path planning (NOPP) algorithm. Solutions constructed by NOPP always have each cell visited exactly once by one UAV, while satisfying all other constraints described in the MIP formulation (described in Sect. 3.2). The relation between solutions from different formulations are illustrated in Fig. 4.

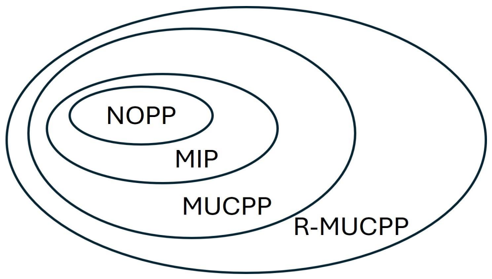

Fig. 4: To solve the MIP, we construct a feasible solution using NOPP and develop a lower bound using R-MUCPP, and then prove that the solution found by NOPP is near or equal to the lower bound.

The NOPP algorithm consists of running four sub-procedures (or phases) sequentially for each UAV. Relevant notations are summarized in Table 4. For a given UAV i, Phase 1 is executed after the initial settings, then the H-value (defined in Sect. 4.2.3) is calculated. If the H-value is odd, Phase 2 is invoked; otherwise, Phase 2 is omitted for this UAV. Subsequently, the number of remaining P moves (i.e., the NP value which is updated in Phases 1 and 2) is checked. If it exceeds zero and is an even number, Phase 3 is directly initiated, and if it is an odd number, then it is incremented to the next even number before Phase 3 is initiated. Finally, if NP is zero, Phase 3 is omitted for this UAV. For the last UAV (i.e., $i = q$), after Phase 3 (if initiated at

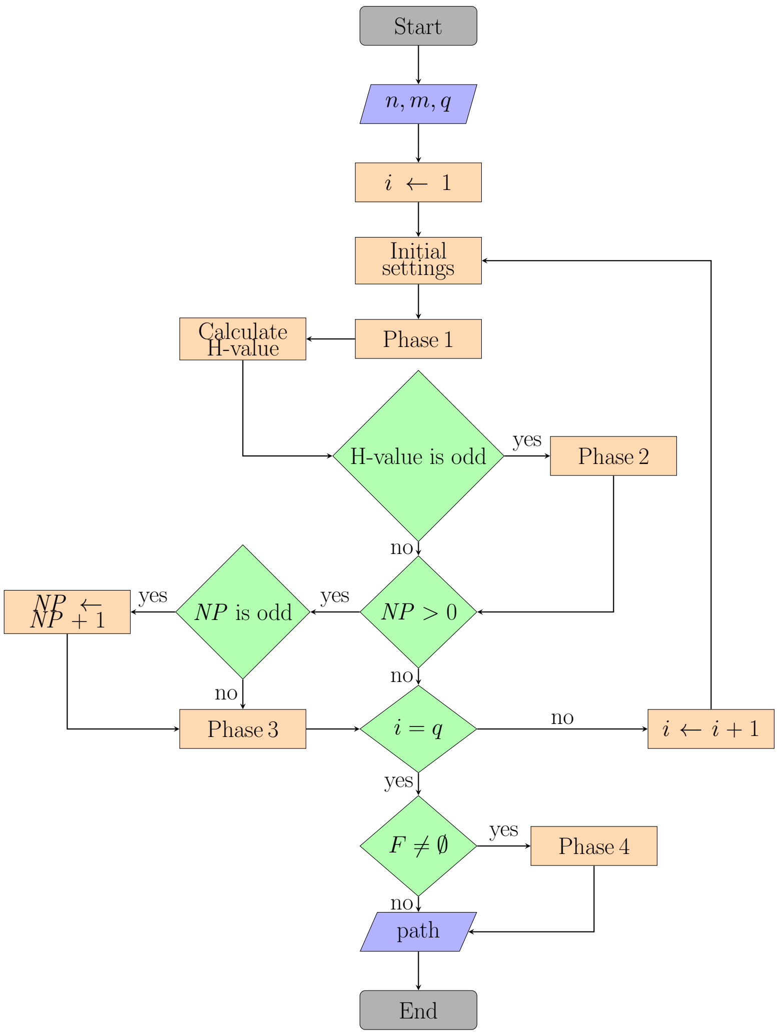

Fig. 5: The flowchart of the NOPP algorithm.

all), Phase 4 will be executed only if the set F is not empty. This phase ensures that all uncovered cells at this point will be covered by the last UAV. The process flow of the NOPP algorithm is illustrated in Fig. 14.

The detailed algorithmic procedures presented below are for cases with $q \leq n$, which account for most situations in practice, that is, fewer UAVs than the number of columns in the grid. A proof of the solution's near optimality is provided in Appendix A. For cases with $q > n$, we can decompose the main problem into multiple subproblems with $q \leq n$, and apply the proposed algorithm to each subproblem with no loss of efficiency (see proof in Appendix B).

4.2.1 Initial settings

Initial values of the variables involved in the algorithm are set as follows: A and NP are assigned $\lceil \frac{nm}{q} \rceil$ and $\lceil \frac{nm}{q} \rceil - n$, respectively (derived from Proposition 2). Additionally, the variable $\bar{O}$ is assigned either True or False based on the parity of NP. Furthermore, the starting point SP for the i^{th} UAV, where $\forall i \in \{1, 2, \ldots, q\}$, is at the coordinate $(1, m - q + i)$. Lastly, the set F encompasses all cells within the $n \times m$ search area, except for those that have already been covered.

4.2.2 Phase one

In this phase, the UAV's path (a sequence of cell coordinates covered by the UAV) starts from the starting point determined in the Sect. 4.2.1. In each iteration, the algorithm evaluates the feasibility of the D, S, and U moves in order, and selects the first feasible move. In addition to being unvisited, a cell must satisfy the following conditions to be a feasible destination for the next move: (1) The UAV cannot move to a cell whose

Y-coordinate is greater than that of the starting point, referred as *Y-coordinate condition* in the rest of the paper; (2) the total number of P moves (the sum of D and U moves) made by a UAV must be less than or equal to $\lceil \frac{nm}{q} \rceil - n$, based on Proposition 2; and (3) if $\bar{O}$ is True, the maximum count of U moves is constrained to be one less than the count of D moves. The pseudo code for Phase 1 is given in Algorithm 1.

Algorithm 1 Phase one of NOPP

```
1:  **function** phaseOne($SP, NP, F, A, \bar{O}$)
2:  Initialize $path$ and $path^*$ to be empty sequence, $NU \leftarrow 0$, $ND \leftarrow 0$, $C \leftarrow SP$
3:  $s_2 \leftarrow$ the second element (Y-coordinate) of $SP$, $path \leftarrow path \cup^* (C)$, and $F \leftarrow F \backslash \{C\}$
4:  **for** 1 through $A$ **do**
5:      **if** $(c_x, c_y - 1) \in F$ and $NP > 0$ **then**
6:          $C \leftarrow (c_x, c_y - 1)$
7:          $path \leftarrow path \cup^* (C)$, $F \leftarrow F \backslash \{C\}$, $path^* \leftarrow path^* \cup^*$ (“D”)
8:          $NP \leftarrow NP - 1$, $ND \leftarrow ND + 1$
9:      **else if** $(c_x + 1, c_y) \in F$ **then**
10:         $C \leftarrow (c_x + 1, c_y)$
11:         $path \leftarrow path \cup^* (C)$, $F \leftarrow F \backslash \{C\}$, $path^* \leftarrow path^* \cup^*$ (“S”)
12:     **else if** $(c_x, c_y + 1) \in F$ and $NP > 0$ and $NU < ND$ and $c_y + 1 \leq s_2$ **then**
13:         **if** $\bar{O}$ **then**
14:             **if** $NU < ND - 1$ **then**
15:                 $C \leftarrow (c_x, c_y + 1)$
16:                 $path \leftarrow path \cup^* (C)$, $F \leftarrow F \backslash \{C\}$, $path^* \leftarrow path^* \cup^*$ (“U”)
17:                 $NP \leftarrow NP - 1$, $NU \leftarrow NU + 1$
18:         **else**
19:             $C \leftarrow (c_x, c_y + 1)$
20:             $path \leftarrow path \cup^* (C)$, $F \leftarrow F \backslash \{C\}$, $path^* \leftarrow path^* \cup^*$ (“U”)
21:             $NP \leftarrow NP - 1$, $NU \leftarrow NU + 1$
22: **return** $path, path^*, F, NP$
```

4.2.3 Phase two

This phase can be executed only when the $path^*$ (generated by Phase 1) concludes with a sequence of U moves, denoted as U^*. Derived from the structure of Phase 1, a sequence of S moves consistently precedes U^*, termed as S^*. We define the *H-value* as $|S^*| - 1$ when the move prior to S^* is a D move, and as $|S^*|$ otherwise. Furthermore, if the $path^*$ does not conclude with a sequence of U moves, the H-value assumes a value of zero, as depicted in Fig. 6b. If the H-value is an odd number, Phase 2 is implemented, during which the algorithm removes U^* from the $path^*$ and updates $path$, $path^*$, F, and NP—all of which were obtained from Phase 1. Algorithm 2 shows the pseudocode for Phase 2.

Algorithm 2 Phase two of NOPP

```
1: **function** phaseTwo($path, path^*, F, NP$)
2: Initialize $pathCopy \leftarrow path, i \leftarrow 0$
3: $i \leftarrow |path^*| - |U^*|$
4: $path \leftarrow$ the first $i + 1$ elements in $path$, $path^* \leftarrow$ the first $i$ elements in $path^*$
5: $F \leftarrow F \cup$ {elements of $pathCopy$ excluding the first $i + 1$ elements}
6: $NP \leftarrow NP + |U^*|$
7: **return** $path, path^*, F, NP$
```

4.2.4 Phase three

This phase is implemented when the count of remaining P moves (NP) is greater than zero. During this phase, the algorithm traverses the $path^*$ to detect two specific subsequences, (“S”, “S”) and (“U”, “S”), referred to as *BI patterns*. It then assesses the feasibility of inserting a single U move before the second move of the BI patterns. The insertion of a U move is considered feasible when a UAV can move upward to visit a cell that has not been covered previously and satisfies the Y-coordinate condition (as described in Sect 4.2.2). If this insertion is feasible, the algorithm inserts one U move before the second move of the BI patterns as well as one D move after that, and updates $path^*$ accordingly. In other words, ("S", "S") is replaced with ("S", "U", "S", "D"), ("U", "S") is replaced with ("U", "U", "S", "D") in $path^*$. This operation is iterated $NP/2$ times by the algorithm.

It should be mentioned that as per the Phase 3 design, NP must be an even number as the input of this phase. Hence, if NP is an odd number after Phases 1 and 2, it is automatically incremented by one and then passed to Phase 3. This adjustment might lead the operation time to reach LB $+ T_p$. The pseudocode for Phase 3 is presented by Algorithm 3.

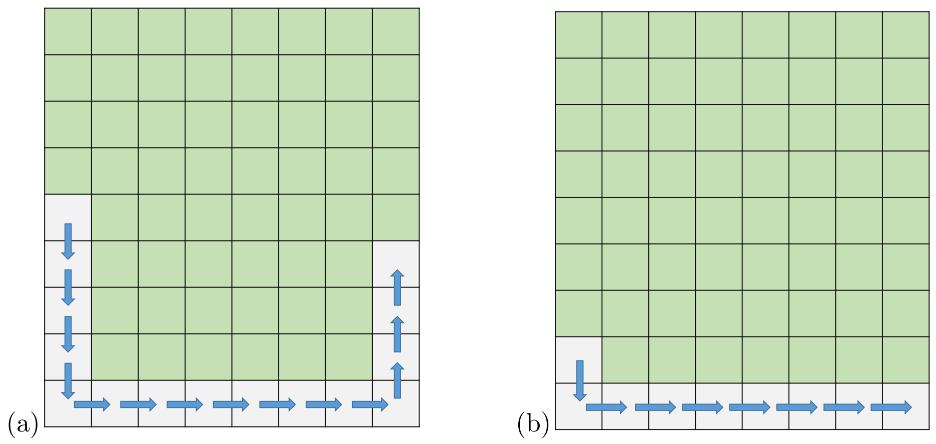

Fig. 6: Example of calculating the H-value for the first UAV after phase 1 in two cases: (a) Due to $|S^*| = 7$ and a D move before S^*, the H-value is 6 ($n = 8,\ m = 9,\ q = 5$); (b) Since there is no U^*, the H-value is 0 ($n = 8,\ m = 9,\ q = 8$).

Algorithm 3 Phase three of NOPP

```
1:  function phaseThree($SP, path, path^*, F, NP$)
2:  Initialize $I \leftarrow 1,\ C \leftarrow SP,\ s_2 \leftarrow$ the second element (Y-coordinate) of $SP$
3:  for 1 through $\frac{NP}{2}$ do
4:      for $i \in \{1, \ldots, |path^*|\}$ do
5:          if $i > I$ and $i^{\text{th}}$ element of $path^*$ is "S" and $(i-1)^{\text{th}}$ element of $path^* \in \{$"U","S"$\}$ then
6:              $C \leftarrow$ the $i^{th}$ element of $path$
7:              if $(c_x, c_y + 1) \in F$ and $c_y + 1 \leq s_2$ then
8:                  Update $path^*$ by placing a "U" and a "D" before and after the $i^{\text{th}}$ element of $path^*$, respectively
9:                  Update $path$ by placing $(c_x, c_y + 1)$ followed by $(c_x + 1, c_y + 1)$ after $C$ in $path$
10:                 $F \leftarrow F \backslash \{(c_x, c_y + 1), (c_x + 1, c_y + 1)\}, NP \leftarrow NP - 2, I \leftarrow i$
11:                 break
12:             else
13:                 continue
14: return $path, path^*, F, NP$
```

4.2.5 Phase four

Due to the Phase 3 design, the algorithm may generate a UAV path where some cells with an X-coordinate equal to n remain uncovered, leaving them for the next UAV to cover. This is not important for other UAVs except for the last one. Since all cells must be covered, the last UAV cannot leave any cells uncovered because there are no other UAVs available to cover them. Therefore, Phase 4 is exclusive to the last UAV and operates only when F is not empty. In other words, the purpose of Phase 4 is to ensure that uncovered cells are covered by the last UAV. Algorithm 4 presents the pseudocode for Phase 4.

Algorithm 4 Phase four of NOPP

```
1: function phaseFour($path, F$)
2: for 1 through $|F|$ do
3:     $C \leftarrow$ the last element of $path$
4:     if $(c_x, c_y + 1) \in F$ then
5:         $path \leftarrow path \cup^* ((c_x, c_y + 1))$
6: return $path$
```

5 Extension to Moore Neighborhood Connectivity

In this section, we demonstrate the effectiveness of our proposed solution when we employ the Moore neighborhood as the selected connectivity type. To do so, we only need to prove that Proposition 1 holds true in the Moore neighborhood connectivity. In the beginning, we introduce several new definitions. Considering Fig. 3, we define an "F move" as a UAV's movement from cell (i, j) to adjacent cells numbered 2 and 8. Likewise, a "B move" refers to the UAV's transition from cell (i, j) to adjacent cells numbered 4 and 6. Furthermore, T_f and T_b denote the time necessary for executing F and B moves, respectively. Let $\bar{D}$ be the distance from cell (i, j) to its neighboring cell numbered 1 (or equivalently 3, 5, and 7); likewise, $\sqrt{2}\bar{D}$ be the distance to neighboring cell numbered 2 (or equivalently 4, 6, and 8). Additionally, we employ the notations $\vec{v}_a, \vec{v}_g$, and $\vec{v}_w$ (as defined in Table 3). Then, we define $\|\vec{v}_a\| = v_a$, $\|\vec{v}_g\| = v_g$, and $\|\vec{v}_w\| = v_w$ and assume that $0 < v_w < v_a$. Before stating the main proposition of this section (Proposition 3), we present a useful inequality as summarized in Lemma 3.

Lemma 3. *In a B move, we have $v_g := \|\vec{v}_a + \vec{v}_w\| \leq \sqrt{2}(v_a - v_w)$.*

Proof. Since the angle between $\vec{v}_g$ and $\vec{v}_w$ in a B move is $\frac{3\pi}{4}$, we have (the inner product of two vectors is represented by $\langle \cdot\ , \cdot \rangle$):

$$\frac{-1}{\sqrt{2}}(\|\vec{v}_a + \vec{v}_w\|)v_w = \frac{-1}{\sqrt{2}}v_g v_w = \langle \vec{v}_g, \vec{v}_w \rangle = \langle \vec{v}_a + \vec{v}_w, \vec{v}_w \rangle = \langle \vec{v}_a, \vec{v}_w \rangle + v_w^2$$
$$\Rightarrow \langle \vec{v}_a, \vec{v}_w \rangle = \frac{-1}{\sqrt{2}}(\|\vec{v}_a + \vec{v}_w\|)v_w - v_w^2$$

The angle θ between $\vec{v}_a$ and $\vec{v}_w$ in a B move is clearly between $\frac{3\pi}{4}$ and π (see Fig. 7), thus we have $-1 \leq \cos\theta \leq \frac{-1}{\sqrt{2}}$. Therefore we can write:

$$-v_a v_w \leq v_a v_w \cos\theta = \langle \vec{v}_a, \vec{v}_w \rangle = \frac{-1}{\sqrt{2}}(\|\vec{v}_a + \vec{v}_w\|)v_w - v_w^2$$
$$\Rightarrow \|\vec{v}_a + \vec{v}_w\| \leq \sqrt{2}(v_a - v_w)$$

□

Now we can state the key proposition of this section.

Proposition 3. *In the Moore neighborhood connectivity where $T_s < T_f < T_p < T_b < T_o$, the minimum time to cover d cells (T), as stated in proposition 1, remains to be*

$$T = \begin{cases} (d-1)T_s & \text{if } d \leq n \\ (n-1)T_s + (d-n)T_p & \text{if } d > n \end{cases}$$

Proof. When $d \leq n$, it is evident that a path with $d - 1$ sequential S moves achieves the shortest time, equal to $(d-1)T_s$. In case where $d > n$, based on Proposition 1, we know that a path with $n - 1$ S moves and $d - n$ P moves, called *VN path*, has the minimum time in Von Neumann connectivity, which is $(n-1)T_s + (d-n)T_p$. Suppose that we can create a faster path than the VN path in the Moore neighborhood connectivity. To do so, we need to replace a P move in the VN path with either an S move or an F move because both T_s and T_f are less than T_p, thus making the path faster.

In the first case, we examine replacing a P move in the VN path with an S move. This action creates a new path that is faster than the VN path, and we refer to it as the *Moore path*. The Moore path consists of n S moves and $d - n - 1$ P moves, resulting in a total path time of $(n)T_s + (d - n - 1)T_p$, which is less than $(n-1)T_s + (d-n)T_p$. Note that the search area is composed of n cells along its length, and employing n S moves within the Moore path results in the UAV exiting the search area. To ensure the UAV remains within the designated search area, it is imperative to substitute a P move in the Moore path with either one B or O move. Given $T_b < T_o$, we update the Moore path by replacing a P move with a B move. After this modification, the Moore path consists of n S moves, one B move, and $d-n-2$ P moves, which takes $(n)T_s + T_b + (d-n-2)T_p$ to cover d cells. To show that the Moore path is faster than the VN path, we have to demonstrate that $(n)T_s + T_b + (d-n-2)T_p < (n-1)T_s + (d-n)T_p$, or equivalently, $T_s + T_b - 2T_p < 0$. By using $T_b = \frac{\sqrt{2}\bar{D}}{\|\vec{v}_a + \vec{v}_w\|}$, we have:

$$\begin{aligned} T_s + T_b - 2T_p &= \frac{\bar{D}}{v_a + v_w} + \frac{\sqrt{2}\bar{D}}{\|\vec{v}_a + \vec{v}_w\|} - \frac{2\bar{D}}{\sqrt{v_a^2 - v_w^2}} \\ &= \frac{\bar{D}((\|\vec{v}_a + \vec{v}_w\|)(\sqrt{v_a^2 - v_w^2}) + \sqrt{2}(v_a + v_w)(\sqrt{v_a^2 - v_w^2}) - 2(v_a + v_w)(\|\vec{v}_a + \vec{v}_w\|))}{(v_a + v_w)(\|\vec{v}_a + \vec{v}_w\|)(\sqrt{v_a^2 - v_w^2})} \end{aligned}$$

Since $\frac{\bar{D}}{(v_a+v_w)(\|\vec{v}_a+\vec{v}_w\|)(\sqrt{v_a^2-v_w^2})} > 0$, we focus on the term $(\|\vec{v}_a+\vec{v}_w\|)(\sqrt{v_a^2 - v_w^2})+\sqrt{2}(v_a+v_w)(\sqrt{v_a^2 - v_w^2})-2(v_a + v_w)(\|\vec{v}_a + \vec{v}_w\|)$ which can be rewritten as:

$$(\|\vec{v}_a + \vec{v}_w\|)(\sqrt{v_a^2 - v_w^2} - 2(v_a + v_w)) + \sqrt{2}(v_a + v_w)(\sqrt{v_a^2 - v_w^2})$$

Since $\sqrt{v_a^2 - v_w^2} - 2(v_a + v_w) < 0$, by using Lemma 3 we have:

$$\begin{aligned} &(\|\vec{v}_a + \vec{v}_w\|)(\sqrt{v_a^2 - v_w^2} - 2(v_a + v_w)) + \sqrt{2}(v_a + v_w)(\sqrt{v_a^2 - v_w^2}) \\ &\geq \sqrt{2}(v_a - v_w)(\sqrt{v_a^2 - v_w^2} - 2(v_a + v_w)) + \sqrt{2}(v_a + v_w)(\sqrt{v_a^2 - v_w^2}) \\ &= \sqrt{2}((v_a - v_w)(\sqrt{v_a^2 - v_w^2} - 2(v_a + v_w)) + (v_a + v_w)(\sqrt{v_a^2 - v_w^2})) \end{aligned}$$

By expressing $v_w = \alpha v_a$ where $\alpha \in (0, 1)$, we can write:

$$\begin{aligned} &(v_a - v_w)(\sqrt{v_a^2 - v_w^2} - 2(v_a + v_w)) + (v_a + v_w)(\sqrt{v_a^2 - v_w^2}) = v_a f(\alpha) \\ &\text{where } f(\alpha) = (1-\alpha)(\sqrt{1-\alpha^2} - 2(1+\alpha)) + (1+\alpha)\sqrt{1-\alpha^2} \end{aligned}$$

The derivative of $f(\alpha)$ can be analytically obtained as $f'(\alpha) = \frac{2\alpha(2\sqrt{1-\alpha^2}-1)}{\sqrt{1-\alpha^2}}$. It is easy to see that for $\alpha \in (0, \frac{\sqrt{3}}{2})$, $f'(\alpha) > 0$, and for $\alpha \in (\frac{\sqrt{3}}{2}, 1)$, $f'(\alpha) < 0$; also, $\alpha = \frac{-\sqrt{3}}{2}, 0, \frac{\sqrt{3}}{2}$ are the only roots of $f'(\alpha)$. Since $f(0) = f(1) = 0$ and $f(\frac{\sqrt{3}}{2}) > 0$, we have $f(\alpha) > 0$ for $\alpha \in (0, 1)$.

Therefore, $T_s + T_b - 2T_p > 0$ and $T_s + T_b > 2T_p$ which leads to a contradiction. It means that the Moore path is slower than the VN path.

In the second case, we consider replacing a P move in the VN path with an F move. Through this action, we create a new path called the *F-Moore path*, which we assume is faster than the VN path. Like the first case, we need to use either a B or O move to keep the UAV inside the search area. Since $T_b < T_o$, we replace a P move in the F-Moore path with a B move and update it. After this update, the F-Moore path includes $n-1$ S moves, one F move, one B move, and $d-n-2$ P moves, which takes $(n-1)T_s + T_f + T_b + (d-n-2)T_p$ to cover d cells. It is clear that $T_f + T_b > T_s + T_b$, and, in the first case, we proved that $T_s + T_b > 2T_p$. Therefore, it is concluded that $T_f + T_b > 2T_p$ which result in $(n-1)T_s + T_f + T_b + (d-n-2)T_p > (n-1)T_s + (d-n)T_p$ and leads to contradiction. It means that the VN path is faster than the F-Moore path.

In the next step, it is necessary to discuss using an O move instead of a B move in the Moore path. Suppose we utilize an O move instead of a B move in the Moore path to maintain the UAV within the search area. Therefore, the Moore path would include n S moves, one O move, and $d-n-2$ P moves. We can observe that the Moore path only utilizes S, O, and P moves. Thus, the Moore path is one of the available paths in the Von Neumann neighborhood connectivity to cover d cells. In Proposition 1, we have proved that the VN path is the fastest path among all available paths in this type of neighborhood connectivity. Therefore, $(n)T_s + T_o + (d-n-2)T_p \geq (n-1)T_s + (d-n)T_p$, or equivalently, $T_s + T_o \geq 2T_p$. Now, it is straightforward to consider using an O move instead of a B move in the F-Moore path to keep the UAV inside the search area. By this change, the F-Moore path consists of $n-1$ S moves, one F move, one O move, and $(d-n-2)$ P moves. Given $T_f > T_s$, it is evident that $T_f + T_o > T_s + T_o$, so we can write $T_f + T_o > 2T_p$, as we have shown that $T_s + T_o \geq 2T_p$. This implies that $(n-1)T_s + T_f + T_o + (d-n-2)T_p > (n-1)T_s + (d-n)T_p$, and therefore, the VN path is still faster than the F-Moore path.

Finally, since it is not possible to create a feasible path faster than the VN path within the Moore neighborhood connectivity, Proposition 1 remains valid even when the Moore neighborhood connectivity is selected.

$\square$

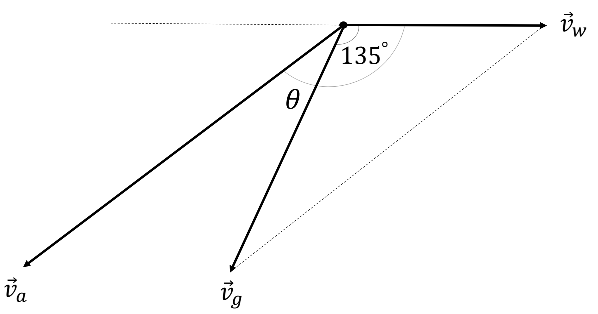

Fig. 7: The relation of three speed vectors, namely, air speed, ground speed and wind speed, in a B move considered in the Moore neighborhood connectivity.

6 Numerical Experiments

The experiments are performed on a PC with a Core i7-7700 processor and 32 GB RAM. The proposed algorithms are implemented in Python 3.7, whereas the MIP models are written in GAMS (version 43.41) and solved by CPLEX (version 22.10).

We first experiment on small test cases to verify the solution obtained from different approaches, and to compare the computational time. Then, four medium-sized instances are solved to showcase the algorithm's performance, and two of them are used to demonstrate how each phase of the NOPP algorithm works. Finally, cases of varying numbers of UAVs and different grid sizes are solved to analyze the effect of problem size on computational complexity.

To determine cell dimensions, recent advancements in AI technology (Martinez-Alpiste et al., 2021) establish target recognition capabilities at distances of 100 meters during flight at a height of 20 meters. Leveraging cutting-edge high-resolution cameras, utilizing a grid cell size of 100×100 meters square becomes a viable choice. The experiments are conducted with UAV and wind speeds set at 20 m/s and 5 m/s, respectively; therefore, transition times between adjacent cells are computed at 4 seconds, 5.16 seconds, and 6.66 seconds for T_s, T_p, and T_o. Codes are available at https://github.com/Sina14KD/CPP-SearchRescue.

6.1 Small-sized cases and Comparison with commercial solver

In this section, we compare the solution quality and computing time between CPLEX and NOPP for small cases of up to 100 cells. The experiments are set up for an $n \times m$ search area utilizing a fleet of q UAVs, and this configuration is represented as a sequence of three values in the first column of Table 5. We impose a time limit of 3600 seconds for each run on CPLEX, and report the upper bound Z (if available) upon the solver's termination.

The first part of Table 5 reports small grids (i.e., up to 20 cells) that can be solved by CPLEX at any allowable choice of q. We can see that for a given grid size, the setting of $q = m$ always gives the easiest instances for CPLEX in terms of computing time, consistent with the simplicity of the optimal solutions for such cases (i.e., each UAV handling a row of the grid). Furthermore, the objective value (Z, optimal when $Z = LB_{\text{CPLEX}}$) found by CPLEX and the lower bound (LB) found by NOPP always agree, which empirically validates our lower bound formula. The second and third parts of the table report cases for which CPLEX struggles to find the optimal solution (and in some cases, even a valid upper bound Z) within the time limit, whereas NOPP always returns a solution in negligible amount of time. For brevity, we only present cases of $q = 2$ in the third part of the table since they are the most challenging cases for NOPP (see Figure 8). Overall, these experiments reveal the inefficiency of directly solving the MIP model using CPLEX, even for fairly small-sized cases. In contrast, the NOPP algorithm takes negligible time to run, and produces solutions belonging to the set $\{\text{LB}, \text{LB} + T_p\}$ across all cases.

6.2 Medium-sized cases and visual explanation

In this section, we investigate medium-sized cases, defined as instances comprising a cell count ranging from 100 to 1000. The outcomes for four specific cases are presented in Table 6, where the initial two cases are used to elucidate the algorithm's functioning. Cases three and four each encompass 1000 cells with different configurations, serving to verify that the algorithm efficiently resolves medium-sized cases within remarkably

Table 5: Results for small-sized test cases

Case (n, m, q)	LB	MIP by CPLEX			NOPP		Relative	Absolute
		Z	LB_{CPLEX}	Time	Z_{NOPP}	Time	gap (%)	gap
(3,3,2)	18.32	18.32	18.32	1.2	23.48	0.00	28.16	5.16
(3,3,3)	8.0	8.0	8.0	0.8	8.0	0.00	0.00	0.00
(4,3,2)	22.32	22.32	22.32	2.08	22.32	0.00	0.00	0.00
(4,3,3)	12.0	12.0	12.0	0.44	12.0	0.00	0.00	0.00
(3,4,2)	23.48	23.48	23.48	2.12	28.64	0.00	21.97	5.16
(3,4,3)	13.16	13.16	13.16	4.88	13.16	0.00	0.00	0.00
(3,4,4)	8.0	8.0	8.0	0.48	8.0	0.00	0.00	0.00
(4,4,2)	32.64	32.64	32.64	49.24	32.64	0.00	0.00	0.00
(4,4,3)	22.32	22.32	22.32	215.2	22.32	0.00	0.00	0.00
(4,4,4)	12.0	12.0	12.0	2.06	12.0	0.00	0.00	0.00
(4,5,2)	42.96	42.96	42.96	201.21	42.96	0.00	0.00	0.00
(4,5,3)	27.48	27.48	27.48	471.3	27.48	0.00	0.00	0.00
(4,5,4)	17.16	17.16	17.16	157.3	17.16	0.00	0.00	0.00
(4,5,5)	12.0	12.0	12.0	5.75	12.0	0.00	0.00	0.00
(5,4,2)	41.80	41.80	41.80	185.71	46.96	0.00	12.34	5.16
(5,4,3)	26.32	26.32	26.32	153.37	31.48	0.00	19.60	5.16
(5,4,4)	16.0	16.0	16.0	5.29	16.0	0.00	0.00	0.00
(5,5,2)	57.28	57.28	48.58	> 3600	62.44	0.00	9.00	5.16
(5,5,3)	36.64	36.64	32	> 3600	36.64	0.00	0.00	0.00
(5,5,4)	26.32	26.32	24	> 3600	31.48	0.00	19.60	5.16
(5,5,5)	16.0	16.0	16.0	19.03	16.0	0.00	0.00	0.00
(6,5,2)	66.44	66.44	58.66	> 3600	66.44	0.00	0.00	0.00
(5,6,2)	67.60	67.60	60.00	> 3600	67.60	0.00	0.00	0.00
(6,6,2)	81.92	83.28	69.16	>3600	81.92	0.00	0.00	0.00
(6,6,6)	20.0	20.0	20.0	3563.8	20.0	0.00	0.00	0.00
(7,7,2)	116.88	159.86	94.58	>3600	122.04	0.00	4.41	5.16
(7,7,7)	24.0	-	24.0	> 3600	24.0	0.00	0.00	0.00
(8,8,2)	151.84	-	124.00	>3600	151.84	0.00	0.00	0.00
(9,9,2)	197.12	-	158.00	>3600	202.28	0.00	2.62	5.16
(9,10,2)	217.76	-	176.00	>3600	217.76	0.00	0.00	0.00
(10,9,2)	216.60	-	176.00	>3600	216.60	0.00	0.00	0.00
(10,10,2)	242.40	-	196.00	>3600	242.40	0.00	0.00	0.00

LB is obtained from LB formula in Sect. 4.1. Z is the objective value returned by CPLEX upon termination. LB_{CPLEX} is the lower bound of CPLEX, relative gap = $\frac{|LB-Z_{NOPP}|}{LB} \times 100$, and absolute gap = $|LB - Z_{NOPP}|$.

brief computational intervals. Additionally, as expected, the results demonstrate that all solutions belong to the set $\{\text{LB}, \text{LB} + T_p\}$.

Table 6: Results for Medium-sized test cases

Case no.	Size of the	Number	LB	NOPP		Relative	Absolute
	$n \times m$ search area	of UAVs		Z_{NOPP}	Time	gap (%)	gap
1	11×10	3	174.16	179.32	0.00	2.9	5.16
2	13×11	4	166.68	166.68	0.00	0	0
3	25×40	2	2547	2552.16	0.25	< 0.01	5.16
4	50×20	6	779.72	779.72	0.15	0	0

LB is obtained from LB formula in Sect. 4.1. Relative gap = $\frac{|LB-Z_{NOPP}|}{LB} \times 100$, and absolute gap = $|LB - Z_{NOPP}|$.

For the first case (case no. 1 in Table 6) depicted in Fig. 9, the algorithm initially plans a path for UAV number one. Preceding the initiation of Phase 1, according to initial settings, the starting point is designated as $(1, 10 - 3 + 1)$, and the total number of assigned cells (A) and available P moves (NP) are 37 and 26 respectively. In the next step, Phase 1 starts its task and descends from the starting point until reaching $(1, 1)$, where the D move becomes unavailable. Then, it continues by S moves till $(11, 1)$, and since the S move is no longer available, it goes up until $(11, 8)$. Next, the H-value is calculated, resulting in an odd value of 9. Consequently, the algorithm, during Phase 2, removes U^* from the path, leaving a path comprising 7 sequential P moves and 10 consecutive S moves. In the subsequent step, the algorithm checks the residual count of P moves (NP) that is 19 (26 minus 7), and given its odd value, the algorithm automatically applies a one-unit incremental adjustment, setting NP to 20. Then, in accordance with the pattern established during Phase 3, the allocation of these 20 P moves takes place along the remaining path from Phase 2, creating the ultimate path of the first UAV. For the next UAV or UAV number 2, the starting point is $(1, 10-3+2)$, and the number of assigned cells and available P moves remain consistent (37 and 26, respectively). Subsequently, in Phase 1, the algorithm constructs a path following the pattern of this phase, which concludes at cell $(11, 9)$. Then, The algorithm calculates the H-value, and since it is 4 (an even number), Phase 2 is omitted. In the next step, given that NP has an even value of 12 (26 minus 14), Phase 3 accomplishes its designated task by placing 12 P moves into the path. Regarding the last UAV or UAV number 3, the algorithm first executes Phases 1, 2, and 3 sequentially, and then, as the last UAV has four phases, the algorithm checks the set of uncovered cells (F). Finally, since there is no uncovered cell, Phase 4 is not implemented.

Figure 10 illustrates the algorithm's performance corresponding to the second case (case no. 2 in Table 6), where, for all four UAVs, phases 1, 2, and 3 operate similarly to the first case. However, unlike the first case, the $\bar{O}$ is true (because the initial value of NP is 23), and therefore, after Phase 1 the number of U moves is one less than D ones. Also, after implementing Phase 3 for the last UAV, set F is not empty and includes cell (13,11). Consequently, during Phase 4, the algorithm employs a U move to cover cell $(13, 11)$.

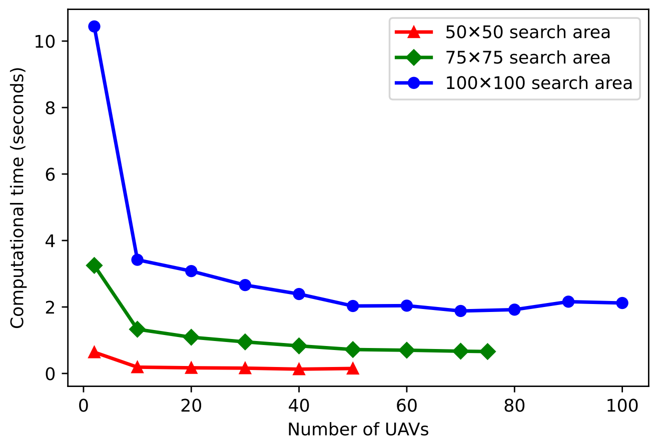

Fig. 8: Impact of the number of UAVs on the computational time.

6.3 Large-sized cases and sensitivity analysis

In this section, the results of the algorithm's performance on large-sized cases, which range from 1000 to 10,000 cells, are presented in Table 7. It is important to note that the results in this section carry greater significance as they reflect the algorithm's efficiency in handling cases that closely resemble real-world scenarios. According to the findings, the algorithm demonstrates a very good performance in large-sized cases, and generates solutions achieving an optimality gap of 0.00%, all within a reasonable timeframe. Also, as expected, all solutions belong to the set $\{\text{LB}, \text{LB} + T_p\}$.

In the following step, we examine the impact of varying the number of UAVs on computational time, as depicted in Fig. 8. This analysis shows some aspects of the algorithm's behavior. Firstly, the maximum computational time is observed when there are only two UAVs. Secondly, while the addition of UAVs does increase problem complexity, it initially accelerates algorithm performance. Thirdly, after a certain point, altering the number of UAVs exhibits no substantial influence on computational time.

Figure 11b illustrates how varying dimensions within a rectangular search area affect computational time. Seven cases with different dimensions are presented, each containing 10,000 cells. The results demonstrate that the algorithm requires less computational time in search areas with smaller length and greater width. Furthermore, it is reaffirmed that across all cases, employing two UAVs consistently demands more computational time. Considering the battery limitation of the UAVs, each operation time has an upper bound. Given one hour as the upper bound of the operation time, Figure 11a shows the minimum number of UAVs required to cover the cases in Table 7 in less than one hour. In this regard, it is noteworthy to mention that the LB formula can provide a very good estimation for calculating the minimum number of UAVs required for an operation with a determined time.

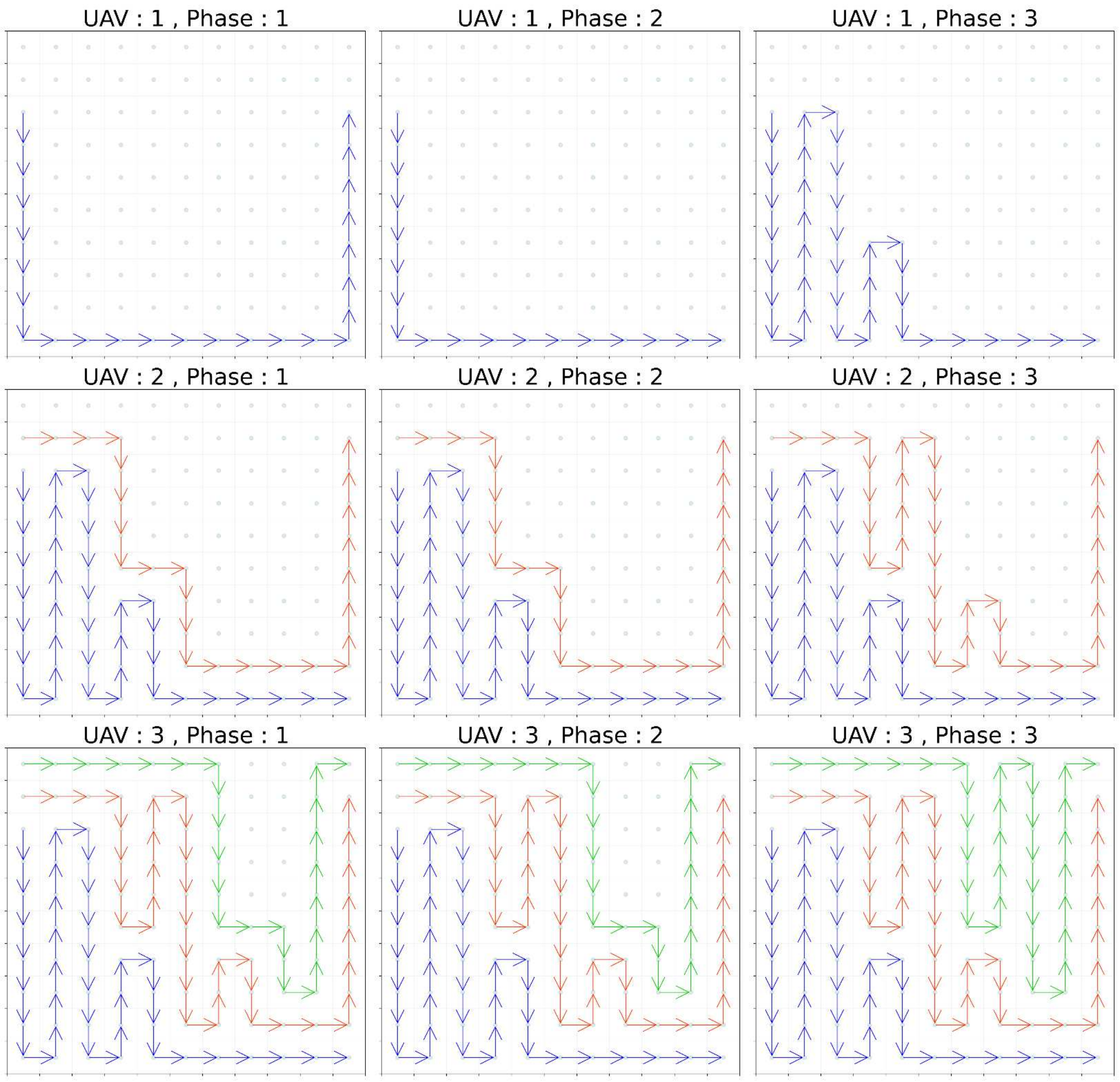

Fig. 9: Demonstration of the algorithm's phases for 3 UAVs in 11 × 10 search area (Case no. 1 in Table 6.)

7 Conclusion and Future Work

Using UAVs for search and rescue comes with its own challenges such as the risk of collision and environmental factors. To deal with these challenges, we developed a path-planning algorithm to enable a homogeneous fleet of UAVs to efficiently search a target area in windy conditions. We first proposed a mixed-integer programming model to formulate the problem. Then, to design a practical solution approach, we investigated a special case of the problem where the search area is a rectangular grid. In this context, we presented a mathematically validated formula for calculating the problem's lower bound. Next, we proposed the NOPP algorithm that consistently yields feasible solutions, achieving either the lower bound or approaching it closely.

We conducted a series of experiments encompassing scenarios of varying search area sizes: small, medium, and large cases. The outcomes of these experiments consistently reveal the algorithm's remarkable proficiency in promptly generating feasible solutions, even when confronted with a large case containing up to 10,000 cells. Also, as indicated by the results, augmenting the problem's complexity by adding more UAVs not only fails to yield a significant impact on the algorithm's speed but, in some cases, can actually lead to an increase in its speed. Furthermore, the results demonstrate that modifying the dimensions of a search area while maintaining

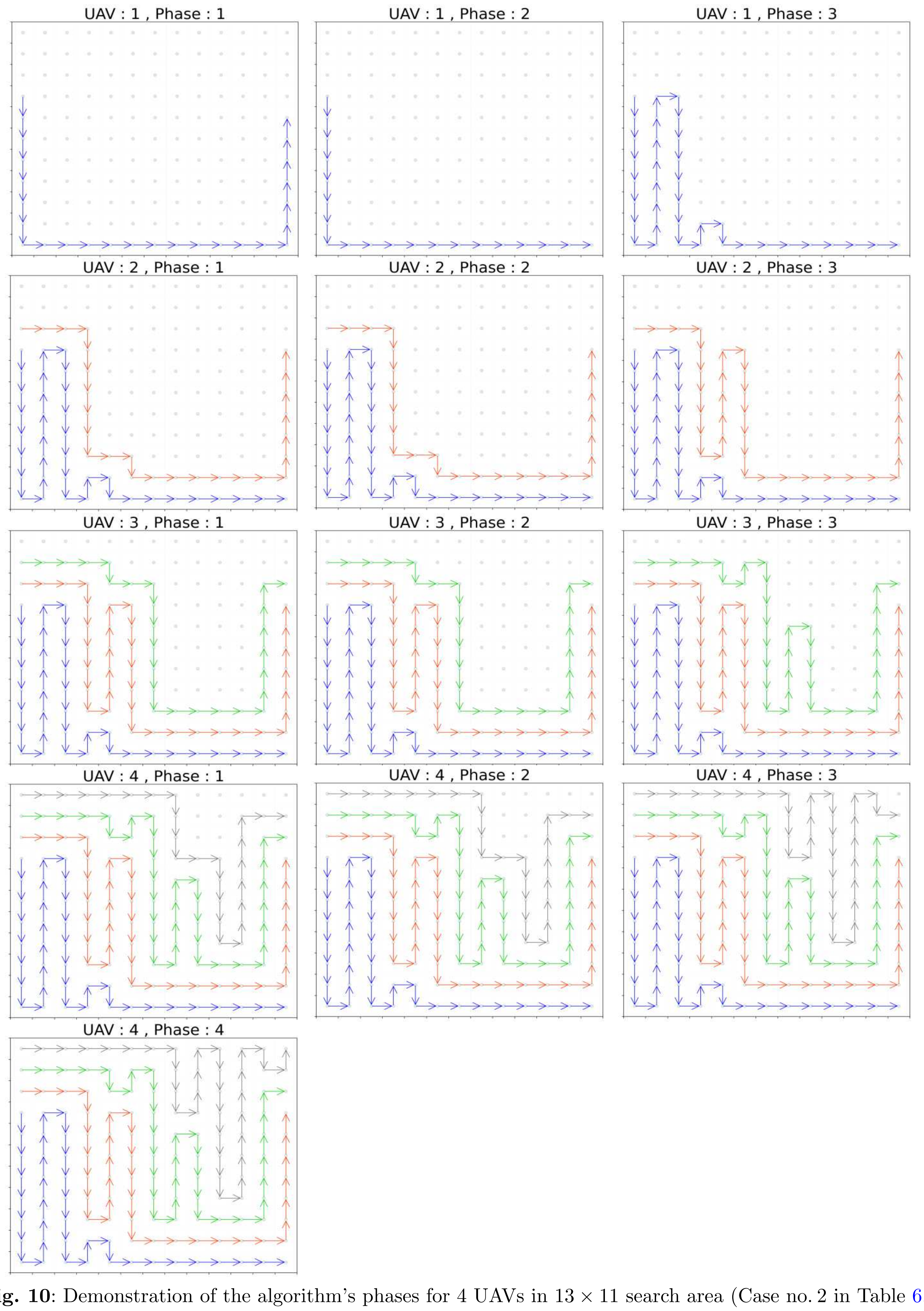

Fig. 10: Demonstration of the algorithm's phases for 4 UAVs in 13×11 search area (Case no. 2 in Table 6.)

Table 7: Results for large-sized test cases

Case no.	Size of the $n \times m$ search area	Number of UAVs	LB	NOPP Z_{NOPP}	NOPP Time	Relative gap (%)	Absolute gap
1	50×50	2	6388.00	6388.00	0.81	0.00	0.00
2	50×75	2	9613.00	9613.00	1.71	0.00	0.00
3	75×50	2	9584.00	9584.00	2.35	0.00	0.00
4	75×75	2	14424.08	14429.24	2.92	< 0.01	5.16
5	75×100	2	19259.00	19264.16	5.37	< 0.01	5.16
6	100×75	2	19230.00	19230.00	6.34	0.00	0.00
7	100×100	2	25680.0	25680.0	10.44	0.00	0.00

LB is obtained from LB formula in Sect. 4.1. Relative gap = $\frac{|LB - Z_{NOPP}|}{LB} \times 100$, and absolute gap = $|LB - Z_{NOPP}|$.

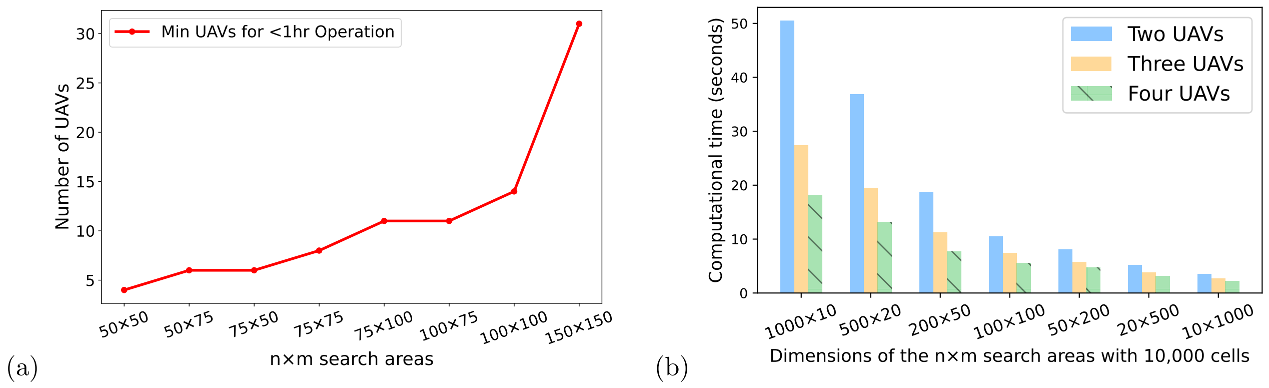

Fig. 11: (a) Minimum UAV number for operations under one hour; (b) Impact of different search area dimensions on computational time.

a constant cell count significantly influences the computational time. In summation, our findings validate the efficacy of the proposed algorithm as a dependable tool for search and rescue operations. Its ability to efficiently devise feasible, optimal, or near-optimal solutions within a reasonable timeframe, particularly in the context of large-scale scenarios, underscores its potential value to search and rescue teams.

Several aspects of the present work can be extended. First, the mathematical framework devised for formulating the lower-bound calculation of the problem can be applied in scenarios involving heterogeneous UAV fleets with unequal speeds. Second, polygonal shapes can be considered as an alternative to rectangular shapes. Our primary focus in this research has been on search and rescue, leading us to concentrate solely on operation time while neglecting other aspects. Future studies can broaden the scope to include factors like battery consumption, landing location, varying wind field and non-uniform distribution of cellular importance, etc., to suit for more application scenarios.

Acknowledgements

The research is supported by the National Science Foundation (NSF) under grant CMMI 1944068. The authors would like to acknowledge the constructive comments from two anonymous reviewers, which helped to improve the quality of this paper.

References

Ahmed, G., Sheltami, T., Mahmoud, A., & Yasar, A. (2023). Energy-efficient uavs coverage path planning approach. *CMES-Computer Modeling in Engineering & Sciences*, *136*(3).

Ai, B., Jia, M., Xu, H., Xu, J., Wen, Z., Li, B., & Zhang, D. (2021). Coverage path planning for maritime search and rescue using reinforcement learning. *Ocean Engineering*, *241*. https://doi.org/10.1016/j.oceaneng.2021.110098

Bähnemann, R., Lawrance, N., Chung, J. J., Pantic, M., Siegwart, R., & Nieto, J. (2021). Revisiting boustrophedon coverage path planning as a generalized traveling salesman problem. *Field and Service Robotics: Results of the 12th International Conference*, 277–290.

Balampanis, F., Maza, I., & Ollero, A. (2017). Spiral-like coverage path planning for multiple heterogeneous uas operating in coastal regions. *2017 International Conference on Unmanned Aircraft Systems, ICUAS 2017*. https://doi.org/10.1109/ICUAS.2017.7991461

Barrientos, A., Colorado, J., Cerro, J. D., 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. *Journal of Field Robotics*, *28*. https://doi.org/10.1002/rob.20403

Bouzid, Y., Bestaoui, Y., & Siguerdidjane, H. (2017). Quadrotor-uav optimal coverage path planning in cluttered environment with a limited onboard energy. *IEEE International Conference on Intelligent Robots and Systems*, *2017-September*. https://doi.org/10.1109/IROS.2017.8202264

Cabreira, T. M., Ferreira, P. R., Franco, C. D., & Buttazzo, G. C. (2019). Grid-based coverage path planning with minimum energy over irregular-shaped areas with uavs. *2019 International Conference on Unmanned Aircraft Systems, ICUAS 2019*. https://doi.org/10.1109/ICUAS.2019.8797937

Cabreira, T. M., Brisolara, L. B., & Ferreira, P. R. (2019). Survey on coverage path planning with unmanned aerial vehicles. *Drones*, *3*(1), 4.

Cho, S. W., Park, J. H., Park, H. J., & Kim, S. (2022). Multi-uav coverage path planning based on hexagonal grid decomposition in maritime search and rescue. *Mathematics*, *10*. https://doi.org/10.3390/math10010083

Cho, S., Park, H. J., Lee, H., Shim, D. H., & Kim, S. Y. (2021). Coverage path planning for multiple unmanned aerial vehicles in maritime search and rescue operations. *Computers and Industrial Engineering*, *161*. https://doi.org/10.1016/j.cie.2021.107612

Choset, H. (2000). Coverage of known spaces: The boustrophedon cellular decomposition. *Autonomous Robots*, *9*. https://doi.org/10.1023/A:1008958800904

Choset, H. (2001). Coverage for robotics–a survey of recent results. *Annals of mathematics and artificial intelligence*, *31*, 113–126.

Coombes, M., Fletcher, T., Chen, W. H., & Liu, C. (2018). Optimal polygon decomposition for uav survey coverage path planning in wind. *Sensors (Switzerland)*, *18*. https://doi.org/10.3390/s18072132

Di Franco, C., & Buttazzo, G. (2015). Energy-aware coverage path planning of uavs. *2015 IEEE international conference on autonomous robot systems and competitions*, 111–117.

Di Franco, C., & Buttazzo, G. (2016). Coverage path planning for uavs photogrammetry with energy and resolution constraints. *Journal of Intelligent & Robotic Systems*, *83*, 445–462.

Forsmo, E. J., Grøtli, E. I., Fossen, T. I., & Johansen, T. A. (2013). Optimal search mission with unmanned aerial vehicles using mixed integer linear programming. *2013 International conference on unmanned aircraft systems (ICUAS)*, 253–259.

Gianfelice, M., Aboshosha, H., & Ghazal, T. (2022). Real-time wind predictions for safe drone flights in toronto. *Results in Engineering*, *15*, 100534. https://doi.org/10.1016/j.rineng.2022.100534

Jiao, Y. S., Wang, X. M., Chen, H., & Li, Y. (2010). Research on the coverage path planning of uavs for polygon areas. *Proceedings of the 2010 5th IEEE Conference on Industrial Electronics and Applications, ICIEA 2010*. https://doi.org/10.1109/ICIEA.2010.5514816

Kong, C. S., Peng, N. A., & Rekleitis, I. (2006). Distributed coverage with multi-robot system. *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, 2423–2429.

Latombe, J.-C. (1991). *Exact cell decomposition*. https://doi.org/10.1007/978-1-4615-4022-9_5

Li, Y., Chen, H., Er, M. J., & Wang, X. (2011). Coverage path planning for uavs based on enhanced exact cellular decomposition method. *Mechatronics*, *21*, 876–885. https://doi.org/10.1016/j.mechatronics.2010.10.009

Liu, Y. (2019). An optimization-driven dynamic vehicle routing algorithm for on-demand meal delivery using drones. *Computers & Operations Research*, *111*, 1–20.

Lyu, M., Zhao, Y., Huang, C., & Huang, H. (2023). Unmanned aerial vehicles for search and rescue: A survey. *Remote Sensing*, *15*(13), 3266.

Martinez-Alpiste, I., Golcarenarenji, G., Wang, Q., & Alcaraz-Calero, J. M. (2021). Search and rescue operation using uavs: A case study. *Expert Systems with Applications*, *178*, 114937.

Maza, I., & Ollero, A. (2007). Multiple uav cooperative searching operation using polygon area decomposition and efficient coverage algorithms. In *Distributed autonomous robotic systems 6* (pp. 221–230). Springer.

Nam, L. H., Huang, L., Li, X. J., & Xu, J. F. (2016). An approach for coverage path planning for uavs. *2016 IEEE 14th International Workshop on Advanced Motion Control, AMC 2016*. https://doi.org/10.1109/AMC.2016.7496385

Park, Y., Ko, C. S., & Moon, I. (2024). Unmanned aerial vehicle variable radius set covering problem for emergency wireless network. *Computers & Operations Research*, *170*, 106765.

Song, H., Yu, J., Qiu, J., Sun, Z., Lang, K., Luo, Q., Shen, Y., & Wang, Y. (2022). Multi-uav disaster environment coverage planning with limited-endurance. *Proceedings - IEEE International Conference on Robotics and Automation*. https://doi.org/10.1109/ICRA46639.2022.9812201

Valente, J., Sanz, D., Cerro, J. D., Barrientos, A., & de Frutos, M. Á. (2013). Near-optimal coverage trajectories for image mosaicing using a mini quad-rotor over irregular-shaped fields. *Precision Agriculture*, *14*. https://doi.org/10.1007/s11119-012-9287-0

Zhang, J., & Li, Y. (2023). Collaborative vehicle-drone distribution network optimization for perishable products in the epidemic situation. *Computers & Operations Research*, *149*, 106039.

Appendix A: Complete proof of feasibility and near-optimality for the NOPP algorithm

We start by introducing a foundational framework that forms the basis for the subsequent propositions of this section. We define C_r^k as a class of grid-based shapes, all of which share the following three properties:

1) All shapes possess r columns, and at least one of the columns has k cells (the distance between the baseline and the top line is k cells, as shown in Fig. 12a).
2) Within each column, cells are required to initiate from the top line and remain connected, no disjoint.
3) Every column contains a minimum of one and a maximum of k cells.

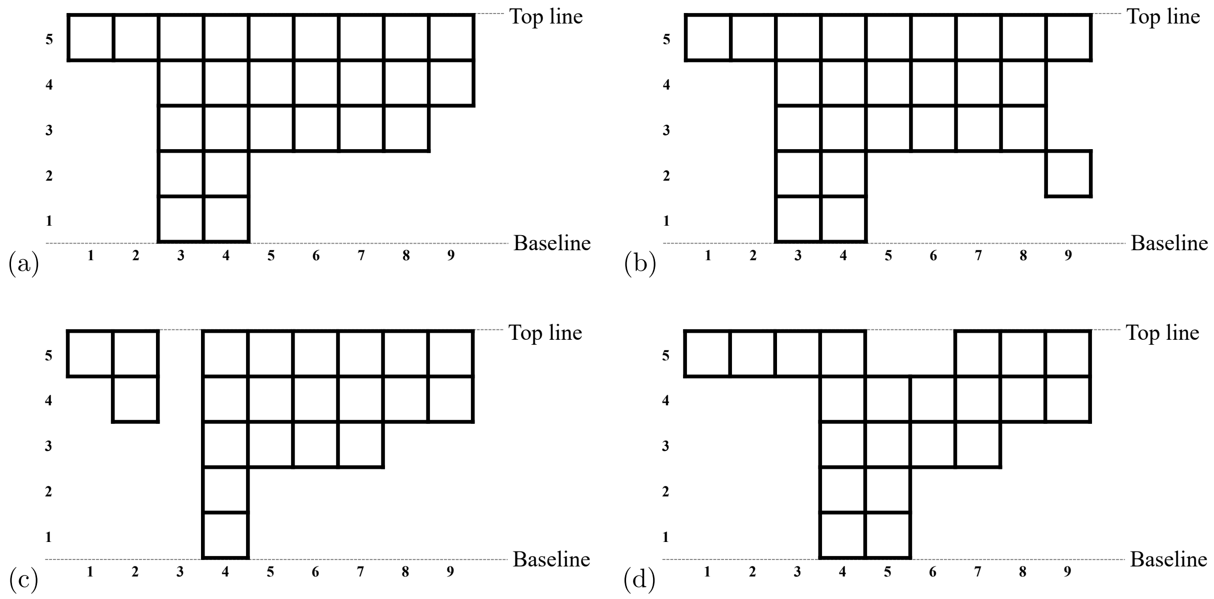

Fig. 12: Valid and invalid examples of C_9^5 shapes: (a) valid; (b) invalid because there is a disjoint in column 9; (c) invalid because column 3 does not have even one cell; (d) invalid because cells in columns 5 and 6 do not initiate from the top line.

In addition, the notations utilized in this section are presented in Table 8 for reference.

Continuing further, according to the definitions in Table 8, we introduce essential terminology employed within the propositions of this section. To begin, for a C_r^k shape, we establish the term *first valley* to denote the valley (as defined in Table 8 as v_t) containing element 1 as well as the term *final valley* to refer to the valley encompassing the element r, and also, the term *middle valley* is designated for other valleys. It should be mentioned that if a valley has both 1 and r, it is considered as the first valley. Furthermore, we introduce the concept of *feasibility feature (FF)*, wherein all valleys in a C_r^k shape are constrained to possess even cardinalities (the cardinality of a valley is equal to its length) except the first and final valleys. The notation "FF-G" is adopted to signify the set G (as defined in Table 8) of a C_r^k shape endowed with the feasibility feature. Moreover, if a C_r^k shape is representable in the form of an FF-G set, it is designated as an *FF-G shape*. For a clearer understanding of defined notations and definitions, consider Figure 12a as an example, depicting an FF-G shape with its corresponding G set equal to $\{\{1,2\},\{3,4\},\{5,6,7,8\},\{9\}\}$. The first valley is $v_1 = \{1,2\}$

Table 8: Algorithm's feasibility proof notations

Notations	
$\bar{I}$	Set of indices pertaining to the columns of the C_r^k shape, $\{1, 2, 3, ..., r\}$
p^i	Set of cells in column i, $\{(i, k), (i, k-1), \dots\}$ for $i \in \bar{I}$, is arranged sequentially and referred to as "pit i"
v_t	Subset of $\bar{I}$ refered to as valley of length t, $\{i_1, \dots, i_t\}$ for $i_k \in \bar{I}$ $(k = 1, \dots, t)$, where: • Indices are sequentially ascending • If $t > 1$, then $\forall i, j \in v_t and\ i \neq j, \lvert p^i\rvert = \lvert p^j\rvert$ • If $i_1 \neq 1$,then $\lvert p^{i_1-1}\rvert \neq \lvert p^{i_1}\rvert$ • If $i_t \neq n$, then $\lvert p^{i_t}\rvert \neq \lvert p^{i_t+1}\rvert$
$d(v_t)$	Depth of an t-length valley calculated as $\forall i \in v_t, d(v_t) = \lvert p^i\rvert$
G	A partition of $\bar{I}$ such that: • All its members are valleys • Elements of each valley are greater than any element of the preceding valley

and to find its depth, we need to check the pits (as defined in Table 8 as p^i) with indices 1 and 2. For the pit with index 1, we have $p^1 = \{(1, 5)\}$, and for the pit with index 2, $p^2 = \{(2, 5)\}$. So, the depth of the first valley is $d(v_1) = |p^1| = |p^2| = 1$. The middle valleys are $v_2 = \{3, 4\}$ and $v_3 = \{5, 6, 7, 8\}$. For v_2, we have to consider pits with index 3 and 4, So $p^3 = \{(3, 5), (3, 4), (3, 3), (3, 2), (3, 1)\}$, and $p^4 = \{(4, 5), (4, 4), (4, 3), (4, 2)(4, 1)\}$; then we have, $d(v_2) = |p^3| = |p^4| = 5$. For v_3, pits with indices $5, 6, 7, 8$ should be considered. So, we have $p^5 = \{(5, 5), (5, 4), (5, 3)\}$, $p^6 = \{(6, 5), (6, 4), (6, 3)\}$, $p^7 = \{(7, 5), (7, 4), (7, 3)\}$, and $p^8 = \{(8, 5), (8, 4), (8, 3)\}$ as well as $d(v_3) = |p^5| = |p^6| = |p^7| = |p^8| = 3$. The final valley is $v_4 = \{9\}$ in which $p^9 = \{(9, 5), (9, 4)\}$ and $d(v_4) = |p^9| = 2$. It should be added that the length of valleys v_1 (the first valley), v_2, v_3, and v_4 (the final valley) are $2, 2, 4$, and 1 respectively. Since the length of the middle valleys (v_2 and v_3) is even, we have an FF-G shape. Now, the following propositions can be stated.

Proposition 4. *There is a feasible path to cover all cells within an FF-G shape.*

Proof. To prove the proposition, it is necessary to consider the following cases for the valleys of set G:

I) All valleys are even-length:
Given that the length of each valley is an even number, they can be divided into one or more pairs of pit indices. For each pair of pit indices, consider the pair of pits corresponding to it. To cover each pair of pits, we can start from the first cell of the first pit (pit with the lower index) and use D moves to reach the end cell. Following this, with a single S move, we can transit to the final cell of the next pit, and subsequently cover all cells in this pit using U moves. Finally, by employing S moves, we can establish connectivity across all pairs.

II) All valleys are even-length except for the first valley:
Initially, omit 1 (index of the first column) from the first valley. As a result, all valleys become even-length, thereby allowing us to use the pattern of case one. Finally, we can cover the cells in p^1 by starting from the last cell and using U moves, and also it can be connected to the next pit by an S move.

III) All valleys are even-length except for the final valley:
Initially, disregard r (index of the last column) from the last valley. Consequently, all valleys become even-length, enabling us to apply the pattern of case one. At the end, cells in p^r can be covered by starting from the first cell and employing D moves. It must be mentioned that an S move makes a connection between p^{r-1} and p^r.

IV) All valleys are even-length except for the first and final valleys:
This case can be effectively resolved by combining cases two and three.

In addition, it is crucial to highlight that when the depth of a valley equals one, the associated pits only need S moves to be covered and connected to other pits. Therefore, it is consistently achievable to create a feasible path for covering an FF-G shape. □

Before proceeding, we need to define two types of FF-G shapes that will be used in propositions 5 and 6. These types are:

- FF-G1: This refers to FF-G shapes with just one valley. For example, a rectangle is an FF-G1 shape.
- FF-G2: This refers to FF-G shapes where the first valley has a depth of one, and the depth of middle valleys is greater than one. For instance, Figure 12a shows an FF-G2 shape.

Also, since the feasibility and optimality of the proposed algorithm are obvious for $q = m$, for the next propositions, we suppose that $q < m$ and $q \leq n$.

Proposition 5. *Upon completion of Phases 1, 2 and 3 of the NOPP algorithm which result in a path for the i^{th} UAV ($i = 1, 2, \ldots, q-1$), available cells for the $(i+1)^{th}$ UAV form an FF-G2 shape.*

Proof. In the beginning, let us state that the available cells for the i^{th} UAV ($i = 1, 2, \ldots, q$) are those uncovered cells that satisfy the Y-coordinate condition (as explained in Sect. 4.2.2). In this proof, we employ g_i (for $i = 1, ..., |G|$) to denote the i^{th} valley (member) of the set G, and we also use g'_i to represent the i^{th} valley after completing Phases 1 and 2. Additionally, we assume that the depth of a valley can be temporarily zero.

First, we consider the FF-G1 shape that is related to the first UAV. Based on the algorithm design, the first UAV starts from the cell $(1, m-q+1)$. Therefore, we can say that the available cells for the first UAV make a C_n^{m-q+1} shape. Since this shape has only one valley ($G = \{\{1, 2, \ldots, n\}\}$), it is an FF-G1 shape.

After Phases 1 and 2, the two following cases can occur for the single valley of the shape (g_1):

I) It is broken down into two valleys (g_α and g_β) that are:
- $g_\alpha = \{1\}$, and $d(g_\alpha) = 0$
- $g_\beta = \{2, \ldots, n\}$, and $d(g_\beta) = d(g_1) - 1$

II) It is broken down into three valleys (g_α, g_β, and g_γ) that are:
- $g_\alpha = \{1\}$, and $d(g_\alpha) = 0$
- $g_\beta = \{2, \ldots, n-1\}$, and $d(g_\beta) = d(g_1) - 1$
- $g_\gamma = \{n\}$, and $d(g_\gamma) = d(g_1) - k$ (for $k = 2, \ldots, d(g_1)$)

Phase 2 guarantees that $|g_\beta|$ is even in the second case. Also, since Phase 3 considers a pair of pits in each valley from the lowest index, the length of the middle valleys is even after implementing this phase. We know that, for the next UAV (the second UAV), the starting point is $(1, m-q+2)$, so the top line for this UAV moves up one cell above the top line of the previous UAV (the first UAV). Therefore, we can add one unit to the depth of the valleys and finally, we have an FF-G2 shape for the second UAV.

Now we have an FF-G2 shape. Initially, disregarding the first and final valleys, the following cases demonstrate how the middle valleys are modified when Phases 1 and 2 are applied. Therefore, these cases for g_i (for $i \in \{2, \ldots, |G|-1\}$) are:

I) If $d(g_{i-1}) < d(g_i) < d(g_{i+1})$, then $|g'_i| = |g_i|$ and $d(g'_i) = d(g_i) - 1$
II) If $d(g_{i-1}) < d(g_i)$ and $d(g_i) > d(g_{i+1})$, then $|g'_i| = |g_i| - 2$ and $d(g'_i) = d(g_i) - 1$
III) If $d(g_{i-1}) > d(g_i) > d(g_{i+1})$, then $|g'_i| = |g_i|$ and $d(g'_i) = d(g_i) - 1$
IV) If $d(g_{i-1}) > d(g_i)$ and $d(g_i) < d(g_{i+1})$, then $|g'_i| = |g_i| + 2$ and $d(g'_i) = d(g_i) - 1$

For the next step, we consider the first valley. Since the first valley is a one-depth valley, there is only one case:

I) $|g'_1| = |g_1| + 1$ and $d(g'_1) = d(g_1) - 1 = 0$

In the following, the final valley, $g_{|G|}$, should be evaluated in some cases. After implementing Phases 1 and 2, there are two cases for the final valley:

I) It is broken into separate valleys (g_a and g_b). So, we have the following sub-cases:
  1) if d($g_{|G|}$) <d($g_{|G|-1}$) then we have:
     - $|g_a| = |g_{|G|}|$, d(g_a)=d($g_{|G|}$)-1
     - $|g_b| = 1$, $g_b = \{n\}$, $d(g_b)$=$d(g_{|G|}) - k$, for $k = 2, 3, \ldots, d(g_{|G|})$
  2) if d($g_{|G|}$) >d($g_{|G|-1}$) then we have:
     - $|g_a| = |g_{|G|}| - 2$, d(g_a)=d($g_{|G|}$)-1
     - $|g_b| = 1$, $g_b = \{n\}$, $d(g_b)$=$d(g_{|G|}) - k$, for $k = 2, 3, \ldots, d(g_{|G|})$

  Phase 2 guarantees that $|g_a|$ is always even.

II) It is not broken into separate valleys, then we have two sub-cases for $g'_{|G|}$:
  1) if d($g_{|G|}$) <d($g_{|G|-1}$)then we have:
     - $|g'_{|G|}| = |g_{|G|}| + 1$, d($g'_{|G|}$)=d($g_{|G|}$)-1
  2) if d($g_{|G|}$) >d($g_{|G|-1}$)then we have:
     - $|g'_{|G|}| = |g_{|G|}| - 1$, d($g'_{|G|}$)=d($g_{|G|}$)-1

After these changes, it is evident that the middle valleys have even lengths following the execution of Phases 1 and 2. As Phase 3 of the algorithm operates on pairs of pits, the parity of the length of middle valleys remains unchanged. For the next UAV (the third UAV), we shift the top line up by one cell, so all the depth of valleys increases by one unit. Consequently, the depth of the first valley is equal to one, and the remaining shape for the third UAV is an FF-G2 shape. Finally, we can see that this pattern will be continued for the next UAVs, and therefore, available cells for i^{th} UAV ($i = 2, \ldots, q$) make an FF-G2 shape.

□

Now, we can prove the feasibility of the generated solutions by the proposed algorithm.

Proposition 6. *The NOPP algorithm generates feasible solutions.*

Proof. According to Proposition 5, the last remaining shape for the final UAV is an FF-G2 shape. Since an FF-G2 shape is a type of FF-G shape, it can be covered by the four phases of the proposed algorithm, as proved in Proposition 4. □

To better understand Proposition 6, Figure 13 illustrates an example featuring five UAVs within a 13 × 13 search area. The proposed algorithm starts generating a path for the first UAV in a rectangle, forming an FF-G1 shape. According to Proposition 5, subsequent shapes for the following UAVs are designated as FF-G2 shapes. Consequently, an FF-G2 shape remains for the fifth and last UAV, which can be covered within four phases of the algorithm.

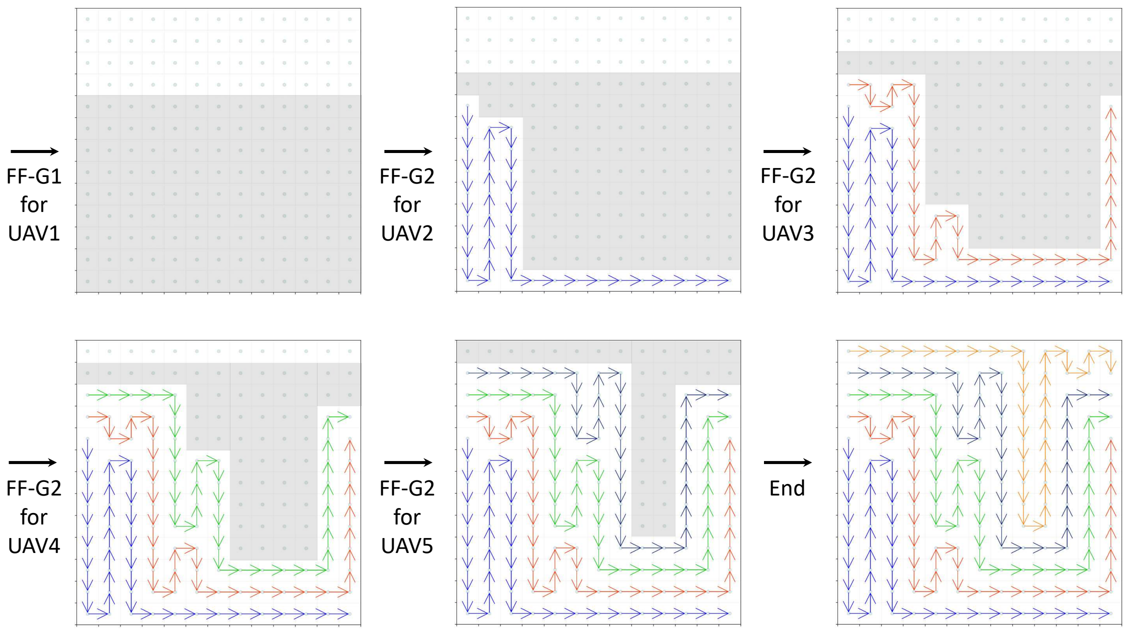

Fig. 13: Example of the feasibility of the algorithm's solutions, as proved in Proposition 6 ($n = 13$, $m = 13$, $q = 5$). All gray shapes are C_{13}^{9} shapes in this example.

By the next proposition, we prove the lower bound and upper bound of the algorithm's solution.

Proposition 7. *The solutions generated by the NOPP algorithm have objective values in the set* $\{LB, LB+T_p\}$.

Proof. Let $A = \lceil \frac{nm}{q} \rceil$ denote the upper bound on the allocation of cells to a UAV required to achieve the LB (as used in Sect. 4.2.1). Based on the proposed algorithm's design, three following cases may occur.

1) The first $q - 1$ UAVs cover either A or $A + 1$ (by using an extra P move as explained in Sect. 4.2.4) cells. As a result, the last UAV is left with fewer than $A + 1$ cells to cover. Consequently, the operation time can be either LB or $LB + T_p$.
2) In this case, we have a UAV, denoted by i^{th} UAV ($1 < i < q$), that covers less than A cells. This situation arises when the number of cells that meet the Y-coordinate condition (as detailed in Sect. 4.2.2) remaining for the i^{th} UAV is less than A, and the i^{th} UAV cover these cells. In this case, UAVs numbered from 1 to $i - 1$ have covered either A or $A + 1$ cells, while UAVs numbered from $i + 1$ to q are required to cover n cells. Since n is consistently less than $\lceil \frac{nm}{q} \rceil$, the resulting operation time is either LB or $LB + T_p$.
3) This case is a sub-state of the previous case. According to the second case, the number of cells left for i^{th} UAV ($1 < i < q$) is less than A. However, in this case, the i^{th} UAV covers all of those remaining cells except some of them whose X-coordinate is n. This situation arises due to the algorithm's phase 3, which considers a pair of pits at each step. In this case, UAVs from 1 to $i - 1$ have covered either A or $A + 1$ cells, and UAV

$i+1$ has a maximum of $n+m-q$ cells to cover. Since $m-q \leq (m-q)\frac{n}{q}$, this leads to $n+m-q \leq \frac{nm}{q} \leq \lceil \frac{nm}{q} \rceil$. If $i \leq q-2$, UAVs from $i+2$ to q should cover n cells. As a result, the operation time is either LB or $LB+T_p$.

□

Appendix B: Addressing cases with $q > n$

In this section, we propose a sub-area decomposition (SD) algorithm to decompose the main $n \times m$ search area into some sub-areas, which can be handled by the NOPP algorithm. The SD algorithm includes two main phases that are *Separation phase* and *Final sub-area phase.*

Separation phase: In this phase, the SD algorithm divides the main $n \times m$ search area into two sub-areas, called down sub-area and up sub-area. Down sub-area is an $n \times m_1$ rectangular area with n UAVs (where $m_1 = \lceil \frac{mn}{q} \rceil$), and the up sub-area is an $n \times (m - m_1)$ rectangular area with $q - n$ UAVs. By using the lower bound formula (described in Sect. 4.1) for the down sub-area we have:

$$(n-1)T_s + (\lceil \frac{m_1 n}{n} \rceil - n)T_p = (n-1)T_s + (\lceil \frac{mn}{q} \rceil - n)T_p$$

So, the lower bound of the down sub-area is equal to the lower bound of the main $n \times m$ search area, and therefore, by implementing the NOPP algorithm in the down sub-area the objective values of the solution belongs to the set $\{LB, LB+T_p\}$ (as proved in Proposition 7). In the last step of this phase, the SD algorithm runs the NOPP algorithm for the down sub-area to cover it. Finally, we show that this division of the main search area into down and up sub-areas is possible by proving that $m_1 < m$ in the following proposition.

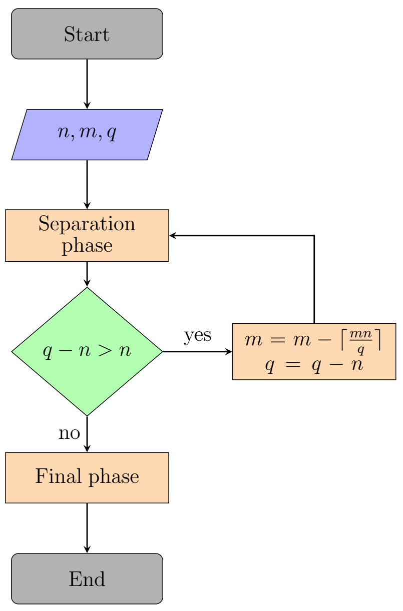

Fig. 14: The flowchart of the sub-area decomposition algorithm.

Proposition 8. *Let m, n and q be natural numbers having the relation $n < q \leq m$, then we have $\lceil \frac{mn}{q} \rceil < m$.*

Proof. By considering $n \leq q-1$ we have:

$$n \leq q-1 \xrightarrow{\times m} nm \leq qm - m \xrightarrow{\times \frac{1}{q}} \frac{nm}{q} \leq m - \frac{m}{q}$$

Since $\frac{m}{q} \geq 1$, we have $m - \frac{m}{q} \leq m-1$; therefore, $\frac{nm}{q} \leq m-1$ and hence $\lceil \frac{mn}{q} \rceil \leq m-1 < m$.

□

After the separation phase, the SD algorithm focuses on the up sub-area (uncovered sub-area) left from the separation phase. To consider more precisely the up sub-area, first, we show that the number of UAVs

$(q-n)$ is less than or equal to the number of cells along the Y-axis of the sub-area $(m - \lceil \frac{mn}{q} \rceil)$, that is, Assumption 1 (described in Sect. 3.1) holds for the up sub-area. To do this, we use the following proposition.

Proposition 9. *Let m, n and q be natural numbers having the relation $n < q \leq m$, then we have $q - n \leq m - \lceil \frac{mn}{q} \rceil$.*

Proof. Since $q \leq m$, there are $k' \in \mathbb{N}$ and $r' \in \mathbb{N} \cup \{0\}$ such that $m = qk' + r'$ so we have:

$$m - \lceil \frac{mn}{q} \rceil = qk' + r' - \lceil \frac{(qk' + r')n}{q} \rceil = qk' + r' - \lceil k'n + r'(\frac{n}{q}) \rceil$$

we know that $\frac{n}{q} < 1$ so:

$$k'n + r'(\frac{n}{q}) < k'n + r' \Rightarrow \lceil k'n + r'(\frac{n}{q}) \rceil \leq k'n + r'$$

and therefore, we can write:

$$qk' + r' - \lceil k'n + r'(\frac{n}{q}) \rceil \geq qk' + r' - (k'n + r') = k'(q-n)$$

we know $k'(q-n) \geq q - n$; hence $qk' + r' - \lceil \frac{(qk'+r')n}{q} \rceil \geq q - n$. $\square$

Now, we need to evaluate the lower bound of the up sub-area. We show that the lower bound of the up sub-area (i.e., $(n-1)T_s + (\lceil \frac{(m-m_1)n}{q-n} \rceil - n)T_p$), is less than or equal to the lower bound of the main search area (i.e., $(n-1)T_s + (\lceil \frac{mn}{q} \rceil - n)T_p$). To do this, it is sufficient to show that $\frac{(m-m_1)n}{q-n} \leq \frac{mn}{q}$, as stated in Proposition 10.

Proposition 10. *Let m, n and q be natural numbers having the relation $n < q \leq m$, and let $m_1 = \lceil \frac{mn}{q} \rceil$, then we have $\frac{(m-m_1)n}{q-n} \leq \frac{mn}{q}$.*

Proof. We need to show the validity of the inequality $q(m - m_1)n \leq mn(q-n)$, so we can write:

$$qmn - qm_1 n \leq qmn - mn^2 \Rightarrow mn^2 \leq qm_1 n \xrightarrow{\times \frac{1}{n}} mn \leq qm_1$$
$$\Rightarrow \frac{mn}{q} \leq m_1$$

Since $\frac{mn}{q} \leq m_1$ is valid, we conclude the validity of the inequality $\frac{(m-m_1)n}{q-n} \leq \frac{mn}{q}$. $\square$

After considering the up sub-area, we can explain the next step of the SD algorithm. Following the separation phase, the SD algorithm checks the inequality $q - n > n$. If this inequality is true, it means that the number of UAVs is greater than n in the up sub-area. So, the SD algorithm considers the up sub-area as the main search area and goes to the separation phase again. If $q - n < n$ is false, the SD algorithm goes to the final phase, which is explained in the following paragraph.

Final phase: In this phase, the SD algorithm implements the NOPP algorithm in the uncovered sub-area which is left from one or more runs of the separation phase.

To clarify the performance of the SD algorithm, we consider a 4×13 search area with 10 UAVs ($n = 4$, $m = 13$, $q = 10$) as an example (see Fig. 15). The SD algorithm first implements the separation phase, and therefore, the down sub-area ($4 \times \lceil \frac{13 \times 4}{10} \rceil$ area with 4 UAVs) and the up sub-area ($4 \times (13 - 6)$ area with 6 UAVs) will be made. Then, the SD algorithm runs the NOPP algorithm for the down sub-area, which covers all cells in this sub-area. In the next step, since $q - n = 10 - 4 = 6$ is greater than $n = 4$, the SD algorithm runs the separation phase in the up sub-area (the uncovered sub-area). Therefore, two sub-areas will be made; first $4 \times \lceil \frac{7 \times 4}{6} \rceil$ area with 4 UAVs (new down sub-area), and second, $4 \times (7 - 5)$ area with 2 UAVs (new up sub-area). The NOPP algorithm is implemented for the new down sub-area and covers cells inside it. Next, $q - n = 6 - 4 = 2$ is less than $n = 4$, so the SD algorithm go to the final phase in which the NOPP algorithm is run for the new up sub-area (4×2 area). At this point, all cells are covered. It should be mentioned that, as we have already shown and proved, the lower bound of the sub-areas generated by each run of the separation phase is less than or equal to the lower bound of the main search area (LB). Therefore, it is clear that the final solution of the SD algorithm belongs to the set $\{LB, LB + T_p\}$.

Finally, it is worth noting that although the SD algorithm cannot cluster starting points together in cases when $q > n$, in real-world large-scale scenarios where $q \leq n$ and $q \leq m$, the NOPP algorithm naturally put all starting points next to each other, which effectively eases the deployment of the UAV fleet.

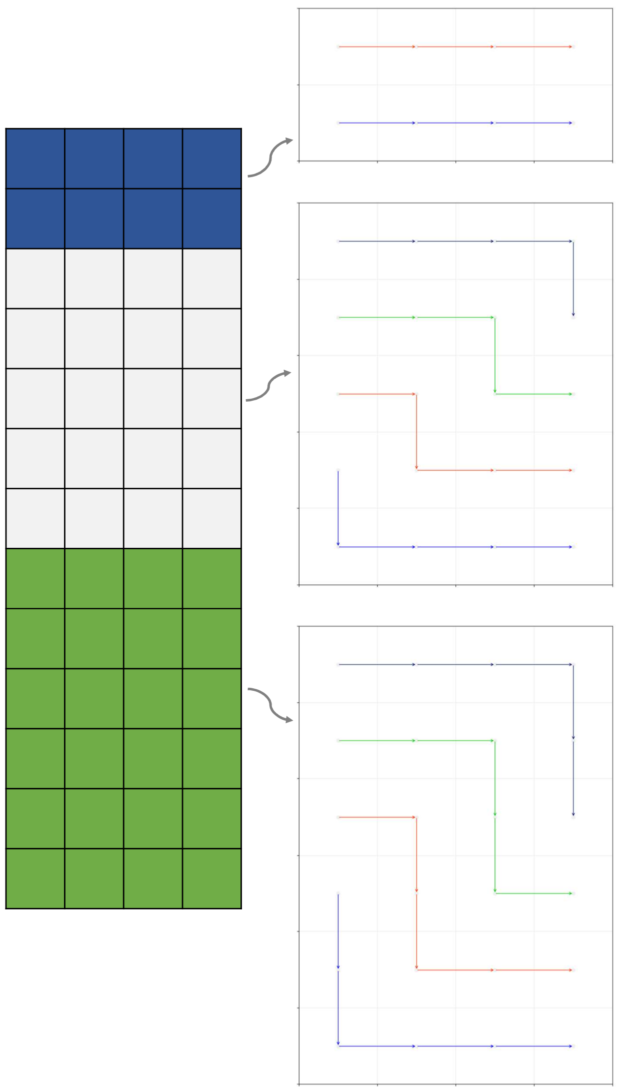

Fig. 15: Sub-areas generated by the SD algorithm for 4×13 search area with 10 UAVs. By assuming $T_s = 10$ (s), and $T_p = 20$ (s), the lower bounds of the 4×6 sub-area with 4 UAVs, 4×5 sub-area with 4 UAVs, and 4×2 sub-area with 2 UAVs are 70 (s), 50 (s), 30 (s) respectively. The LB of the main search area (4×13 search area with 10 UAVs) is 70 (s) which is equal to the maximum of the sub-areas lower bound (max $\{70,50,30\}$). This indicates that the SD algorithm decomposes the main search area without loss of efficiency. Note that the mission time (objective value) for this example is 70 (s) that belongs to $\{LB,\, LB + T_p\}$.