From Docking Station to Docking Station: Completing Tasks in Minimum Time by Cooperative UAV Fleets

Baixin Wan, Feng Shan*, Jianping Huang
School of Computer Science and Engineering
Southeast University, Nanjing, China
*Corresponding author. Email: shanfeng@seu.edu.cn

***Abstract*—As a growth engine for smart cities, the emergence of the low-altitude economy presents significant scheduling challenges for UAV fleets that perform cooperative tasks, such as logistics and infrastructure inspection. This paper focuses on scheduling a fleet of cooperative, homogeneous UAVs to perform a series of tasks distributed along a predefined route between docking stations, with the objective of minimizing the total completion time. The complexity stems from temporal dependencies that affect subsequent tasks and from idle-time constraints, where early-arriving UAVs must wait for collaborators. To tackle this, we propose a novel framework based on a geometric visualization of the scheduling problem, introducing the concept of a "skyline" to create a canonical representation of resource availability over time. A provably optimal schedule is found based on this skyline concept, and computational efficiency heuristics that incorporate a lookahead mechanism is proposed for large-scale, practical applications. Comprehensive simulations demonstrate that our skyline-based methods consistently and significantly outperform conventional baseline algorithms in solution quality.**

***Index Terms*—Multi-UAV, Cooperative Scheduling, Completion Time Minimization, Resource-Constrained Scheduling, Dynamic Programming, Heuristics.**

I. Introduction

The low-altitude economy, driven by Unmanned Aerial Vehicles (UAVs), is emerging as a new driver of growth [1]. Automated UAV docking stations (*e.g.* DJI's Dock [2]) enable drones to land, recharge, and resume missions autonomously, significantly enhancing their operational endurance. Consequently, UAV docking stations are becoming essential infrastructures that support large-scale, automated UAV operations within the *low-altitude economy* [3], which demands new forms of **elastic and collaborative intelligence** [4]. Although UAVs are agile and cost-effective in dense urban environments [5], their significant limitations in payload, energy, and onboard processing make a single UAV insufficient for complex tasks [6], [4]. As a result, swarm collaboration becomes essential, where a fleet of UAVs is coordinated to accomplish missions beyond the capability of any individual agent. **For example**, a group of water-carrying firefighting drones can simultaneously spray water at a tall building to extinguish fires; multiple loudspeaker drones may be deployed to manage emergency events and broadcast important messages to crowds; and a set of base-station drones can provide communication support for a large number of ground devices during public gatherings. In the literature, such collaborative missions are actively studied as complex scheduling challenges, such as UAV service provisioning [7] or correlation-aware task offloading [8]. Therefore, this paper focuses on collaboration tasks that require a number of UAVs to work together.

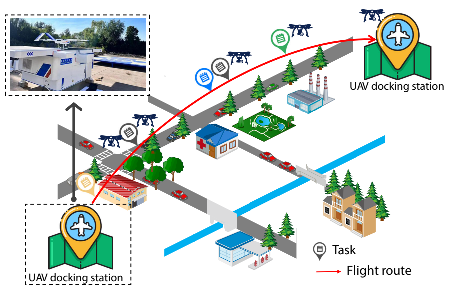


Fig. 1: Scheduling a fleet of cooperative, homogeneous UAVs to complete a series of tasks distributed along a predefined route, with the goal of minimizing the completion time of all tasks. Any UAV can only travel from one docking station to another without returning, and must execute tasks sequentially along its path. Each task requires a specific number of UAVs to be simultaneously available for a given duration, which may cause early-arriving UAVs to remain idle while waiting for collaborators.

This paper addresses a crucial problem in this context: scheduling a fleet of cooperative, homogeneous UAVs to perform a series of tasks distributed along a predefined, energy-efficient route [9], as illustrated in Fig. 1. The route begins at one automated UAV docking station and ends at another, which is a part of a larger smart city UAV scheduling framework discussed in our other work [10]. Each UAV can perform multiple tasks, while each task requires a certain number of UAVs to cooperate over a specified duration. Therefore, the central objective is to coordinate UAV movements and task assignments so that all tasks are completed in the minimum possible time.

Minimizing the completion time of all tasks through coordinated UAV fleet movements and task assignments is highly challenging. **First**, a UAV can only travel from one docking station to another without returning, and must execute tasks sequentially along its path. Since each task requires a certain amount of time, its execution introduces temporal dependencies that affect subsequent tasks. **Second**, each task requires a specific number of UAVs to be simultaneously available for a given duration. This may cause early-arriving UAVs to remain idle while waiting for collaborators, thus how to reduce such idle time is challenging in considering the UAV-to-task assignments. **Finally**, the search space of possible UAV-to-task assignments grows exponentially with the number of tasks and UAVs, rendering brute-force or exhaustive search approaches computationally infeasible [11].

Existing research on multi-agent task allocation, including coalition formation frameworks [12], [13], [14], [15], primarily focuses on high-level resource assignment. In contrast, the problem investigated in this paper emphasizes practical challenges in the low-altitude economy, where a fleet of cooperative UAVs travels from one docking station to another. These challenges include temporal dependencies arising from sequential task execution along the route and idle time caused by waiting for collaborators. This shift in focus calls for a new and specialized methodology.

To this end, we develop a novel scheduling method built upon a geometric visualization of the problem space. Our main contributions are threefold:

- This paper formulates a novel scheduling problem that aims to minimize the completion time of all tasks by a cooperative UAV fleet traveling from one docking station to another. We further provide an equivalent problem transformation that decouples UAV movements from task assignments. This simplification allows us to focus on UAV-to-task assignments without any loss of optimality.
- This paper introduces a novel geometric concept, *skyline*, to represent UAV availability. Building on this, we develop a provably optimal scheduling algorithm, Skyline-Based Exact Dynamic Programming (S-EDP). To address large-scale instances, we further propose *skyline*-inspired heuristics with improved computational efficiency that incorporate a lookahead mechanism.
- We design and conduct a comprehensive set of simulation experiments to validate our framework. The results, evaluated across various task profiles and problem scales, demonstrate significant and consistent performance gains for our proposed algorithms over conventional scheduling heuristics, reducing the mission completion time by up to approximately 10% in our test cases.

The remainder of this paper is organized as follows. Section II reviews the related work. Section III details the system model and problem formulation. Section V presents our S-EDP and its underlying pruning strategies. Section VI introduces the heuristic algorithms for large-scale problems. Section VII provides a comprehensive performance evaluation, and Section VIII concludes the paper.

II. Related Work

A. Multi-UAV Task Allocation and Coalition Formation

In the literature, related work on *coalition formation* shares similarities with our study in terms of theoretical framework. Coalition formation [16] is a classic paradigm in multi-agent systems that investigates how autonomous agents form collaborative groups to execute tasks beyond the capability of any single agent. The literature presents diverse approaches, including cost-based distributed algorithms [13], [17], utility-driven models with resource constraints [14], [18], and more recent game-theoretic formulations tailored for UAV networks [19], [15]. However, a prevailing assumption in this body of work is the emphasis on determining which agents should form a coalition, while the operational logistics of task execution are often oversimplified. Such abstraction overlooks the stringent constraints of real-world applications where operations are confined to predefined corridors. Our work addresses this gap by explicitly incorporating temporal dependencies from sequential task execution and idle time due to waiting for collaborators.

B. Two-dimensional Strip Packing

When visualized geometrically, our scheduling problem is visually similar to the *two-dimensional strip packing problem* [20]. This connection inspired the design of our baseline algorithms; for instance, the best-fit strategy [20], [21] we adopt is a classic heuristic originating from this field. However, the similarity is limited, as fundamental differences in core constraints prevent the direct application of traditional packing algorithms [22]. Classic packing problems deal with indivisible, rigid rectangles that can be placed in arbitrary order. In contrast, our task-rectangles are "divisible" along the UAV index axis and, more importantly, UAVs must execute tasks sequentially, introducing temporal dependencies absent in packing problems. Therefore, a specialized scheduling framework is required.

III. System Model and Problem Formulation

A. System Model

We consider a set of M UAVs, denoted by $\mathcal{U} = \{u_1, u_2, \ldots, u_M\}$, which cooperatively execute a sequence of n tasks, $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$, distributed in a predefined and unidirectional route. Each task $t_i \in \mathcal{T}$ is characterized by a tuple (d_i, m_i), where $d_i > 0$ is its execution duration and $m_i \in \{1, \ldots, M\}$ is the number of UAVs required. The flight time for a UAV to reposition from the location of task t_i to that of t_j is denoted by $f(t_i, t_j)$. A schedule P assigns to each UAV $u_k \in \mathcal{U}$ an ordered sequence of tasks to execute. This assignment must ensure that for each task $t_i \in \mathcal{T}$, the set of assigned UAVs, denoted by $U(t_i) \subseteq \mathcal{U}$, satisfies its resource requirement $|U(t_i)| = m_i$. Also, based on this assignment, the schedule determines a feasible start time $b(t_i)$ and completion time $c(t_i)$ for each task t_i.

TABLE I: Key Notation for Problem Definition

Notation	Definition
$\mathcal{U}$	The set of M UAVs, $\{u_1, \ldots, u_M\}$.
$\mathcal{T}$	The set of n tasks, $\mathcal{T} = \{t_1, \ldots, t_n\}$.
$f(t_i, t_j)$	The flight time from task t_i to t_j.
d_i	The execution duration of task t_i.
m_i	The number of UAVs required by task t_i.
$U(t_i)$	The set of UAVs from $\mathcal{U}$ assigned to task t_i.
$b(t_i)$	The start time of execution assigned to task t_i.
$c(t_i)$	The completion time of task t_i, where $c(t_i) = b(t_i) + d_i$.
$C_{max}(P)$	The completion time of a schedule P, $\max_i\{c(t_i)\}$.

In our problem context, any feasible schedule must adhere to the ordering constraint where UAVs fly along a unidirectional path, which establishes a fixed traversal sequence for all tasks based on their location. Thus, any single UAV must execute its assigned tasks strictly in this predefined order. This leads to the following constraint for any pair of tasks, t_a and t_b, that share a common UAV:

$$c(t_i) + f(t_i, t_j) \leq b(t_j), \forall t_i, t_j \in \mathcal{T}, i < j, U(t_i) \cap U(t_j) \neq \emptyset \tag{1}$$

In addition, the task atomicity constraint, motivated by real-world applications such as aerial surveillance or agricultural spraying where tasks are inherently continuous and uninterruptible, is formalized as follows: once a task t_i begins, the assigned set of UAVs $U(t_i)$ remains fixed, with all UAVs starting and completing *synchronously* at $b(t_i)$ and $c(t_i)$. This can be expressed mathematically as:

$$c(t_i) = b(t_i) + d_i, \forall t_i \in \mathcal{T}, \forall u_k \in U(t_i). \tag{2}$$

A key assumption of this model is that all UAVs are homogeneous, i.e., functionally identical. This reflects practical scenarios such as aerial inspection, surveillance, and logistics, where a uniform fleet simplifies maintenance and scheduling operations.

B. Problem Formulation

Based on the system model, the SR-MUCSP is formally defined as follows:

Definition 1 (SR-MUCSP)**.** *Given a set of M homogeneous UAVs $\mathcal{U}$, a sequence of n tasks $\mathcal{T}$, the Single-Route Multi-UAV Cooperative Scheduling Problem (SR-MUCSP) aims to find an optimal schedule P. This schedule determines the sequence of tasks assigned to each UAV, such that the completion time $C_{max}(P)$ is minimized, while the ordering constraint in Eq. (1) and task atomicity constraint in Eq. (2) are satisfied.*

IV. Problem Transformation

This section transforms the Problem SR-MUCSP into an equivalent logical scheduling problem without considering UAVs' movement. It then introduces a geometric visualization to visualize the problem, laying the groundwork for the algorithm design that follows.

A. Decoupling of Movement Cost

To simplify the model, we decouple the physical flight cost from the scheduling problem. This simplification is based on the optimal flight mode of UAVs. As demonstrated by real-world flight experiments, a practical, energy-optimal flight mode defines the optimal cruise speeds, acceleration/deceleration, and turning strategies to minimize energy consumption for a given route [9]. In our SR-MUCSP, it is therefore reasonable to posit that all UAVs adopt this same optimal flight mode to maximize system-wide efficiency. This results in the movement cost $f(t_a, t_b)$ being the same for any UAV. This decoupling is also robust: even if individual UAV movement costs vary slightly in practice (e.g., due to wind or battery degradation), the assignment plan P derived from our logical model remains highly effective.

Based on this, we can abstract this complex but identical flight behavior into an equivalent average optimal speed v. Therefore, the time required to cover the entire predefined route of total length $\mathcal{L}$, denoted by the constant $\tau = \mathcal{L}/v$, is identical for every UAV. We can express the total time a UAV u_k takes to complete its duties in the physical world, p_k, as the sum of its logical completion time l_k (time spent purely on tasks and waiting) and the constant route time τ.

$$p_k = l_k + \tau. \tag{3}$$

The overall physical completion time, C^p_{max}, is the maximum of these individual completion times. Since τ is a constant shared by all UAVs, it can be factored out of the maximization:

$$C^p_{max} = \max_{u_k \in \mathcal{U}}\{p_k\} = \max_{u_k \in \mathcal{U}}\{l_k + \tau\} = \max_{u_k \in \mathcal{U}}\{l_k\} + \tau. \tag{4}$$

By defining the logical completion time as $C^l_{max} = \max_{k \in \mathcal{U}}\{l_k\}$, we arrive at the direct relationship:

$$C^p_{max} = C^l_{max} + \tau. \tag{5}$$

This derivation proves that a schedule minimizing the logical completion time also minimizes the physical completion time. We can therefore focus on solving the simpler problem, which we term the *Logical SR-MUCSP (LSR-MUCSP)*, where times for inter-task movement are effectively considered zero. The constant τ can simply be added to the final result.

B. Geometric Visualization and State Representation

To facilitate the analysis and solution of the LSR-MUCSP, we introduce a geometric visualization. The scheduling process is visualized on a two-dimensional *scheduling strip*, where the horizontal axis represents time and the vertical axis represents the M available UAV resources (Fig. 2 Left). Within this visualization, each task t_i corresponds to a *composite rectangle* with a width of d_i and a total height of m_i. This composite rectangle is itself composed of m_i left-aligned, unit-height rectangles, which can be placed non-contiguously along the UAV Index axis. A feasible schedule for the LSR-MUCSP is thus visualized as a non-overlapping placement of these composite rectangles within the strip. The objective, in this geometric context, is to minimize the maximum rightmost coordinate of any placed rectangle.

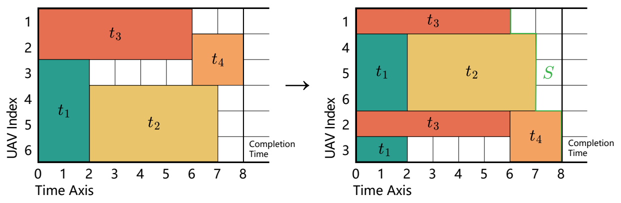


Fig. 2: Geometric visualization of a feasible schedule and its canonical visualization. The schedule is visualized on a 2D strip where the x-axis denotes time and the y-axis denotes the UAV resources (indexed 1 to M). Each rectangle corresponds to a task t. Critically, UAVs are not sorted by fixed IDs but are dynamically re-ordered by their earliest available times. This process forms the unique, monotonic profile S (highlighted in green) that defines the Skyline of the scheduling state.

Given the fixed task sequence $\mathcal{T}$, a natural approach is a sequential process that schedules tasks one by one. In this process, the partial schedule for the first i tasks forms a *scheduling state*, which we formally denote by Σ_i. This state serves as the basis for all subsequent scheduling decisions. To ensure a unique and standardized representation for any scheduling state, we introduce a *canonical visualization* rule: on the UAV Index axis, UAVs are not sorted by their fixed IDs but are dynamically ordered by their earliest available times, l_k, in *ascending* order. This convention ensures that the boundary of available resources forms a unique, monotonically *non-decreasing* profile, which we term the *skyline* (Fig. 2 Right).

Definition 2 (Skyline)**.** *The skyline of a scheduling state Σ_i, denoted by $S(\Sigma_i)$, is a vector of the M earliest available times of the UAVs, sorted in ascending order: $S(\Sigma_i) = (s_1, s_2, \dots, s_M)$, where $s_1 \leq s_2 \leq \dots \leq s_M$.*

A scheduling state Σ_i is thus fully characterized by its skyline $S(\Sigma_i)$ and its task assignment history. The skyline provides a complete summary of resource availability, serving as the basis for all forward-looking scheduling decisions. The assignment history is used only to reconstruct the final schedule after the search is complete. Due to the homogeneity of the UAVs, scheduling decisions can be made based solely on the skyline vector, abstracting away the specific UAV IDs. This abstraction is fundamental to the state-space reduction techniques that follow.

V. Framework for Optimal Scheduling

The previous section established a geometric framework for the LSR-MUCSP, the logical equivalent of the original problem. Building on this foundation, this section introduces pruning techniques, namely state dominance and candidate placement points, to construct an efficient state-space search framework. Several scheduling algorithms are subsequently designed based on this framework.

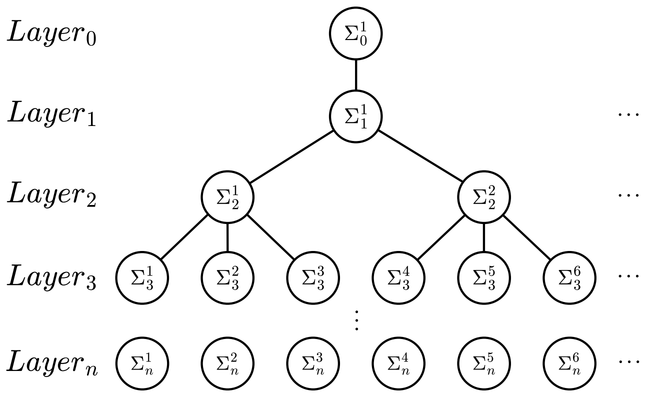


Fig. 3: The state transition graph, which models the scheduling problem within a dynamic programming framework. The graph is structured as a layered tree, where each node Σ_i at Layer i represents a possible scheduling state after the first i tasks have been placed. A directed edge from a state in Layer $i-1$ to a new state in Layer i represents a specific placement decision for task t_i. The entire search process aims to find the optimal state in Layer n.

A. Framework: The State Transition Graph

All subsequent algorithms are built upon a *state transition graph*, a concept rooted in dynamic programming. This graph is a layered *tree* with $n+1$ layers, corresponding to the task sequence $\mathcal{T}$ (Fig. 3). The nodes at layer i are all possible scheduling states $\{\Sigma_i\}$ after scheduling the first i tasks, with Layer 0 containing the single root node Σ_0. Each state is characterized by its skyline $S(\Sigma_i)$ and assignment history. A directed edge connects a parent state Σ_i to a child state Σ_{i+1}, representing a step placement for t_i on the scheduling state of the parent node.

An exhaustive search of this state-space tree is computationally infeasible for two main reasons. First, the number of outgoing edges from any state is effectively infinite, as a task can be placed at countless positions on the scheduling strip. Second, the tree would contain immense redundancy, because different scheduling histories (i.e., different nodes) can lead to equivalent states with identical skylines. Pruning strategies are therefore essential to make the search tractable. The following subsections introduce techniques to eliminate these redundant states and to reduce the number of placements considered, while preserving optimality.

B. Pruning Strategy I: Skyline-Based State Dominance

Our first pruning strategy eliminates redundant states within the same layer of the search tree. Due to the homogeneity of the UAVs, the specific IDs of the UAVs assigned to a task are irrelevant for future decisions. Only their availability times, captured by the skyline. Therefore, whether one state is superior to another can be determined purely by comparing their skylines. This is formalized through a dominance relation shown in Fig. 4.

Definition 3 (State Dominance)**.** *Given two states Σ_A and Σ_B with skylines $S(\Sigma_A) = (a_1, \dots, a_M)$ and $S(\Sigma_B) =$*

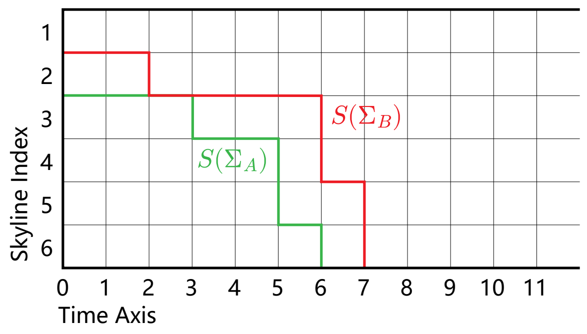


Fig. 4: The State Dominance concept, a key state-space pruning strategy. The figure compares skylines of two states, $S(\Sigma_A)$ and $S(\Sigma_B)$. Σ_A is said to dominate Σ_B (denoted $\Sigma_A \preceq \Sigma_B$) because each of its availability times is less than or equal to the corresponding time in Σ_B. Consequently, any dominanted state, such as that of Σ_B, can be safely pruned without sacrificing optimality, as long as states prepare to schedule the same task.

*$(b_1, \dots, b_M)$ sorted in ascending order, we say that Σ_A **dominates** Σ_B, denoted $\Sigma_A \preceq \Sigma_B$, if and only if $a_j \le b_j$ for all $j \in \{1, \dots, M\}$, a vector relationship which we denote as $S(\Sigma_A) \le S(\Sigma_B)$.*

Geometrically, a dominating state Σ_A offers at least as much, and potentially more, available area for future tasks than Σ_B. The following stronger lemma provides a robust foundation for pruning.

Lemma 1 (Preservation of Dominance)**.** *Let Σ_A and Σ_B be two states at the same layer $i-1$. If $\Sigma_A \preceq \Sigma_B$, then for any child state Σ'_B of Σ_B at layer i, there exists a corresponding child state Σ'_A of Σ_A such that Σ'_A dominates Σ'_B.*

Proof. Let $S(\Sigma_A) = (a_1, \dots, a_M)$ and $S(\Sigma_B) = (b_1, \dots, b_M)$. The premise $\Sigma_A \preceq \Sigma_B$ implies $a_j \le b_j$ for all $j \in \{1, \dots, M\}$.

Consider any child state Σ'_B generated by placing task t_i on a set of m_i rows (indices) $K \subseteq \{1, \dots, M\}$ of the skyline $S(\Sigma_B)$. Let the start time be $s_b = \max_{j\in K}\{b_j\}$. The new availability times for the rows in K become $s_b + d_i$.

We construct Σ'_A by placing t_i on the same set of rows K of $S(\Sigma_A)$. The resulting start time is $s_a = \max_{j\in K}\{a_j\}$. Since $a_j \le b_j$ for all j, it follows that $s_a \le s_b$.

Let us compare the multisets of availability times that form the new skylines.

- For any unselected row $j \notin K$, we have $a_j \le b_j$.
- For any selected row $j \in K$, the new time is $s_a + d_i$, which is less than or equal to $s_b + d_i$.

Every value in the multiset for state A is less than or equal to a corresponding value in the multiset for state B. This element-wise dominance is preserved after sorting to form the new skylines. Thus, $\Sigma'_A \preceq \Sigma'_B$. □

If a state Σ_A dominates another state Σ_B at the same layer, Σ_B can be safely pruned from the search space. This is because any optimal schedule reachable from Σ_B can be matched or improved upon by a corresponding schedule reachable from the state Σ_A. This dramatically reduces the number of states to be explored at each layer.

C. Pruning Strategy II: Reduction of Placement Positions

While state dominance prunes the number of states (nodes), it does not address the infinite number of actions (edges) from each state. For any given state Σ_i, placing the next task t_{i+1} offers infinite possibilities due to the continuous nature of start times. This section introduces a strategy to reduce these infinite placements to a small, finite set of dominant candidates (Fig. 5). A placement of task t_{i+1} on the skyline $S(\Sigma_i) = (s_1, \dots, s_M)$ is defined by the chosen set of m_{i+1} row indices, $K \subseteq \{1, \dots, M\}$. Let Σ_{i+1} be the resulting child state, and we have the following two properties.

Optimality of Left-Alignment. The earliest possible start time for a placement on a set of rows K is $s^*(t_{i+1}) = \max_{k\in K}\{s_k\}$. Any other placement with a start time $s'(t_{i+1}) > s^*(t_{i+1})$ is non-left-aligned. Let Σ_{i+1} be the state generated by the left-aligned placement and Σ'_{i+1} be the state from the non-left-aligned placement. It is evident that $S(\Sigma_{i+1}) \preceq S(\Sigma'_{i+1})$, as the latter unnecessarily delays the new availability times. Thus, we only need to consider left-aligned placements.

Optimality of Downward-Alignment. For any given left-aligned placement using a set of rows K, a "downward-swap" can be applied. This involves replacing a selected row $a \in K$ with an unselected row $b \notin K$ where $a < b$, provided the start time does not increase. As proven in Lemma 1, such a swap results in a new, dominating skyline because our skyline is sorted in ascending order ($a < b \implies s_a \le s_b$). A placement that is irreducible to such swaps must consist of a contiguous block of rows starting at a "corner". Such a placement is downward-aligned and dominates any non-contiguous or non-corner placement.

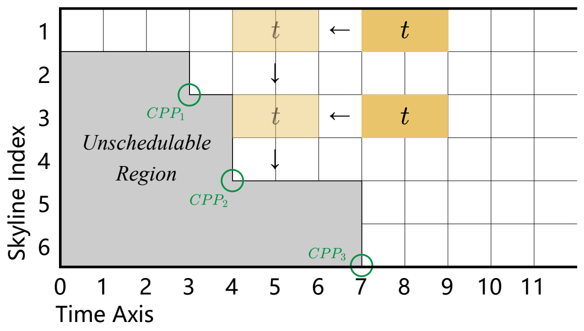


Fig. 5: Mechanism for reducing an arbitrary placement to a Candidate Placement Point (CPP). The figure demonstrates how any arbitrary placement of task t is transformed into an optimal placement at a CPP (green circles) via a two-step process: Left-Alignment(shifting the task to the skyline boundary) and Downward-Alignment(moving it to higher-indexed UAVs without delaying its start time). This reduction is critical because a placement at a CPP is proven to yield a dominating skyline. It allows the algorithm to evaluate only the finite set of CPPs, reducing the search space from infinite to a few candidates without loss of optimality.

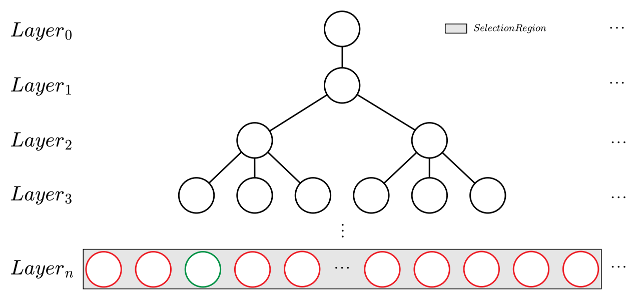


Fig. 6: The state transition and selection graph of S-EDP. The "Selection Region" (gray box) represents a set of candidate states from which only one is chosen to advance while the rest are pruned. In this final step of S-EDP, the algorithm selects the single state (highlighted in green) that yields the minimum completion time ($C_{\max}$). Since the search terminates at this final layer, the selected state's schedule represents the globally optimal solution.

Algorithm 1 Skyline-Based Exact DP (S-EDP)

```
Input: Tasks $\mathcal{T} = \{t_1, \dots, t_n\}$, UAV number $M$.
Output: An optimal scheduling state $\Sigma_{final}$.
  Let $\mathcal{P}_0, \mathcal{P}_1, \dots, \mathcal{P}_n$ be sets of states for each layer.
  $\mathcal{P}_0 \leftarrow \{\Sigma_0\}$;
  for $i = 1$ to $n$ do
      $\mathcal{S}_i \leftarrow \emptyset$;    // Candidate states for the current layer
      for $\Sigma_{parent} \in \mathcal{P}_{i-1}$ do
          $CPPs \leftarrow$ FindCPPs($S(\Sigma_{parent}), t_i$);
          for $cpp \in CPPs$ do
              $\Sigma_{child} \leftarrow$ PlaceTask($\Sigma_{parent}, t_i, cpp$);
              $\mathcal{S}_i$.add($\Sigma_{child}$);
          end for
      end for
      $\mathcal{P}_i \leftarrow$ Prune($\mathcal{S}_i$);
  end for
  $\Sigma_{final} \leftarrow \arg\min_{\Sigma \in \mathcal{P}_n} \{C_{max}(\Sigma)\}$;
  return $\Sigma_{final}$.
```

Definition 4 (Candidate Placement Point, CPP)**.** *Formally, a position for placing task t_{i+1} on the skyline $S(\Sigma_i) = (s_1, \dots, s_M)$ is a Candidate Placement Point (CPP) if and only if the set of m_{i+1} row indices it occupies, K, forms a contiguous block $\{j, j-1, \dots, j-m_{i+1}+1\}$ where the starting index j satisfies the "corner" condition: $s_j < s_{j+1}$ or $j = M$,and the start time of the task $b(t_{i+1}) = \max_{k \in K}\{s_k\}$.*

These two properties prove that for any possible placement, there exists a CPP that generates a dominating or equal skyline. Therefore, the search for a subsequent state can be restricted to considering only placements on CPPs. This reduces the infinite set of possible edges from each state to a finite set of at most $O(M)$ dominant candidates, making the search tractable without loss of optimality.

D. Skyline-Based Exact DP

Skyline-Based Exact DP (S-EDP) is a direct implementation of the state-space search framework, as outlined in Algorithm 1. It constructs the pruned State Transition Graph layer by layer to find a globally optimal solution (Fig. 6). The core of this dynamic programming approach is captured by the following state transition equation. This equation describes the transition process for $\mathcal{P}_i$, the set of skylines corresponding to the mutually non-dominating states after scheduling the first i tasks:

$$\mathcal{P}_i \subseteq \mathcal{S}_i \quad \text{and} \quad \forall S_a, S_b \in \mathcal{P}_i, (a \neq b) \implies S_a \not\preceq S_b. \tag{6}$$

$$\mathcal{S}_i = \begin{cases} \bigcup_{S \in \mathcal{P}_{i-1}} \left(\bigcup_{j \in \mathcal{J}} \{\text{Sort}((v_1, v_2, \dots, v_M))\} \right), & i \geq 1 \\ \{(0, \dots, 0)_{1 \times M}\}, & i = 0, \end{cases} \tag{7}$$

where the set $\mathcal{S}_i$ generated at layer i is pruned to eliminate dominated states, resulting in the non-dominated set $\mathcal{P}_i$. This set $\mathcal{P}_i$ then serves as the basis for generating the next layer's full set, $\mathcal{S}_{i+1}$. The set $\mathcal{J}$ in the state transition equation contains the starting row indices of all Candidate Placement Points (CPPs) and is defined as follows:

$$\mathcal{J} = \{j \in \mathbb{Z} \mid (m_i \leq j \leq M) \wedge (j = M \vee s_j < s_{j+1})\}. \tag{8}$$

Finally, $(v_1, \dots, v_M)$ represents the new, unsorted skyline vector formed after placing a task. Each component v_k is defined by:

$$v_k = \begin{cases} \max\limits_{p=j-m_i+1}^{j} \{s_p\} + d_i, & \text{if } j - m_i + 1 \leq k \leq j \\ s_k, & \text{otherwise.} \end{cases} \tag{9}$$

1) Time Complexity Analysis: The time complexity of S-EDP is primarily driven by the total number of states generated during the search. Our Candidate Placement Positions (CPP) pruning strategy is critical here, as it restricts task placements to contiguous blocks of UAVs. This limits the branching factor at any given state for task t_i to exactly $M - m_i + 1$, which is strictly upper-bounded by M. As the search proceeds for n layers, this establishes a theoretical time complexity upper bound of $O(M^n)$.

In practice, the actual runtime is substantially lower than this pessimistic bound for two main reasons. First, the branching factor is often much smaller than M, reaching this maximum only for single-UAV tasks ($m_i = 1$). Second, and more critically, our skyline dominance pruning strategy eliminates a large, though unpredictable, number of dominated states at each layer, drastically reducing the effective search space.

2) Space Complexity Analysis: The space complexity is determined by the total number of states stored across all n layers of the search graph. The size of the set of mutually non-dominating states at any single layer, $W_{\max} = \max_i |\mathcal{P}_i|$, is theoretically bounded by the length of the longest antichain in the state space. By analogy to Sperner's theorem [23], this theoretical maximum grows exponentially with the number of UAVs M, and can be represented by the central binomial coefficient $\binom{M}{\lfloor M/2 \rfloor}$. This leads to a pessimistic theoretical space complexity bound of $O(n \cdot \binom{M}{\lfloor M/2 \rfloor})$.

However, this upper bound is rarely approached in practice. The combination of our two pruning strategies—Candidate Placement Positions and state dominance—ensures that the effective number of states is typically much smaller than this theoretical maximum. Therefore, the actual space requirement

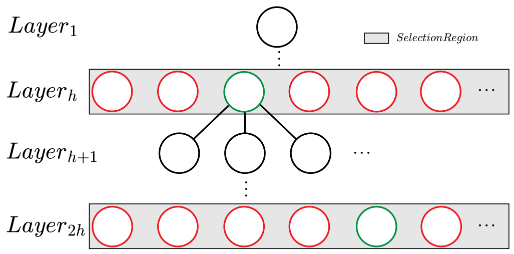


Fig. 7: The state transition and selection graph of S-SDP.The task sequence is divided into segments of a fixed size h. At each segment boundary (e.g., Layer h), a "Selection Region" is formed over the candidate states. Only a single state with the best local completion time(highlighted in green) is selected to initiate the search for the next segment, while all other states (red) are pruned. This inter-segment pruning strategy sacrifices guaranteed optimality for computational tractability.

is often significantly lower, though its precise magnitude is data-dependent and unpredictable a priori.

VI. Heuristic Algorithms

S-EDP guarantees optimality but is computationally infeasible for large-scale problems. This section presents three heuristic algorithms designed to find high-quality solutions efficiently: a segmented dynamic programming approach with a fixed lookahead, a simple greedy algorithm, and a more sophisticated heuristic DP with a dynamic lookahead mechanism.

A. Skyline-Based Segmented DP

S-EDP's exponential complexity renders it infeasible for large-scale problems. A practical approach is Skyline-Based Segmented DP (S-SDP), outlined in Algorithm 2, which divides the task sequence into segments of a fixed size, a, and applies S-EDP to each one. To prevent the exponential growth of states, an aggressive pruning step is performed at each segment boundary (Fig. 7): only a single state (typically the one with the minimum completion time) is selected to serve as the sole starting point for the next segment. This greedy and inter-segment pruning sacrifices global optimality for tractability. If the task sequence is divided into $\lceil n/a \rceil$ segments of size at most a, the algorithm provides an $(\lceil n/a \rceil)$-approximation. The complexity is also significantly reduced; the time complexity becomes $O(\frac{n}{a} \cdot (M^a))$, while the space complexity is reduced to that of a single segment of size a. However, a key limitation of this greedy selection is that it ignores the "shape" of the skyline, which can compromise resource availability for subsequent segments and motivates the design of more advanced heuristics.

B. Skyline-Based Greedy

Skyline-Based Greedy (S-G) algorithm is the most straightforward heuristic. At each step i, it schedules the next task t_i by selecting the CPP that results in the earliest possible completion time. This can be viewed as a special case of S-SDP where the segment length is one ($a = 1$). Although

Algorithm 2 Skyline-Based Segmented DP (S-SDP)

```
Input: Tasks $\mathcal{T} = \{t_1, \dots, t_n\}$, UAV number $M$, Segment size $a$.
Output: A final scheduling state $\Sigma_{final}$.
  $i \leftarrow 0$;
  $\mathcal{P}_{start} \leftarrow \{\Sigma_0\}$;
  while $i < n$ do
      $\mathcal{P}_0 \leftarrow \mathcal{P}_{start}$;    // Initialize the first layer of the segment's DP
      $k \leftarrow 0$;    // Counter for tasks within the current segment
      while $k < a$ and $i + k + 1 \leq n$ do
          $k \leftarrow k + 1$;
          $\mathcal{S}_k \leftarrow \emptyset$;
          for $\Sigma_{parent} \in \mathcal{P}_{k-1}$ do
              $CPPs \leftarrow$ FindCPPs($S(\Sigma_{parent}), t_{i+k}$);
              for $cpp \in CPPs$ do
                  $\Sigma_{child} \leftarrow$ PlaceTask($\Sigma_{parent}, t_{i+k}, cpp$);
                  $\mathcal{S}_k$.add($\Sigma_{child}$);
              end for
          end for
          $\mathcal{P}_k \leftarrow$ Prune($\mathcal{S}_k$);
      end while
      // Inter-segment pruning: select only the best state to proceed
      $\Sigma_{best} \leftarrow \arg\min_{\Sigma \in \mathcal{P}_k} \{C_{max}(\Sigma)\}$;
      $\mathcal{P}_{start} \leftarrow \{\Sigma_{best}\}$;
      $i \leftarrow i + k$;
  end while
  $\Sigma_{final} \leftarrow \mathcal{P}_{start}$.get_first();    // The final remaining state
  return $\Sigma_{final}$.
```

presented as a offline method, the algorithm is inherently an **Online Algorithm**, making it well-suited for dynamic task arrivals. S-G directly maps to an execution pattern where the earliest arriving UAVs are prioritized for the task. Due to its low time complexity, approximately $O(n \cdot M)$, and minimal memory use, this algorithm is effective for rapidly generating solutions.

C. Skyline-Based Heuristic DP

To overcome the short-sightedness of the greedy approach, we designed Skyline-Based Heuristic DP (S-HDP), as outlined in Algorithm 3. This algorithm uses a lookahead window of size a_{max} to build a small DP state-space tree. Instead of greedily committing to a single step, it chooses the most promising intermediate state from the entire lookahead tree based on a holistic evaluation function, $Score(\Sigma)$ (Fig. 8). This score is a weighted sum of three metrics: completion time increment $M_{\Delta C_{max}}$ (ΔC_{max}), waste rate M_{waste} (the blank area included in the skyline divided by the area of the skyline), and profile quality $M_{profile}$ ($S(\Sigma)$'s Coefficient of Variation).

To determine a robust and high-performing weight configuration, we conducted comprehensive computational simulations (Fig. 9) across three primary scenarios. These scenarios are generated by sampling from three distinct task types, where for each task t_i, its duration d_i and required UAVs m_i are drawn from uniform distributions:

- **Normal tasks (t_{normal}):** $d_i \sim U(1, 0.75M)$ and $m_i \sim U(1, 0.75M)$.
- **Time-dominant tasks (t_{TS}):** $d_i \sim U(0.75M, M)$ and $m_i \sim U(1, 0.5M)$.

Algorithm 3 Skyline-Based Heuristic DP (S-HDP)

```
Input: Tasks $\mathcal{T} = \{t_1, \dots, t_n\}$, UAV number $M$, Lookahead size $a_{max}$, Weights $W$.
Output: A final scheduling state $\Sigma_{final}$.
  $\Sigma_{current} \leftarrow \Sigma_0$;
  $i \leftarrow 0$;   // Index of the next task to be scheduled globally
  while $i < n$ do
      $a \leftarrow \min(a_{max}, n-i)$;   // Determine lookahead window size
      // Build a local DP table for the lookahead window
      Let $DP\_layers_0, \dots, DP\_layers_a$ be sets of states.
      $DP\_layers_0 \leftarrow \{\Sigma_{current}\}$;
      for $k = 1$ to $a$ do
          $\mathcal{S}_k^{cand} \leftarrow \emptyset$;
          for $\Sigma_{parent} \in DP\_layers_{k-1}$ do
              $CPPs \leftarrow$ FindCPPs($S(\Sigma_{parent}), t_{i+k}$);
              for $cpp \in CPPs$ do
                  $\Sigma_{child} \leftarrow$ PlaceTask($\Sigma_{parent}, t_{i+k}, cpp$);
                  $\mathcal{S}_k^{cand}$.add($\Sigma_{child}$);
              end for
          end for
          $DP\_layers_k \leftarrow$ Prune($\mathcal{S}_k^{cand}$);
      end for
      // Final segment: greedily minimize completion time
      if $(i + a == n)$ then
          $\Sigma_{best} \leftarrow \arg\min_{\Sigma \in DP\_layers_a}\{C_{max}(\Sigma)\}$;
          $k_{consumed} \leftarrow a$;
      else   // Intermediate segment: use holistic scoring function
          $best\_score \leftarrow \infty$;
          for $k = 1$ to $a$ do
              for $\Sigma \in DP\_layers_k$ do
                  if Score($\Sigma, W$) $< best\_score$ then
                      $best\_score \leftarrow$ Score($\Sigma, W$);
                      $\Sigma_{best} \leftarrow \Sigma$;
                      $k_{consumed} \leftarrow k$;
                  end if
              end for
          end for
      end if
      $\Sigma_{current} \leftarrow \Sigma_{best}$;
      $i \leftarrow i + k_{consumed}$;
  end while
  return $\Sigma_{current}$.
```

- **Resource-dominant tasks (t_{RS}):** $d_i \sim U(1, 0.5M)$ and $m_i \sim U(0.75M, M)$.

The three scenarios are then composed as follows: the **TS** scenario contains 75% t_{TS} and 25% t_{normal} tasks; the **BS** scenario consists entirely of t_{normal} tasks; and the **RS** scenario contains 75% t_{RS} and 25% t_{normal} tasks.

For each scenario, the weights $(W_{C_{\max}}, W_{\text{waste}}, W_{\text{profile}})$ were varied from 0 to 1 in steps of 0.05, subject to their sum being 1. To ensure statistical reliability, 128 independent runs were performed for every valid configuration at each problem scale. To evaluate the quality of each configuration, we define its 'Performance' as the average final completion time (C_{max}) achieved across all runs, which directly aligns itself with the paper's primary goal of minimizing C_{max}. The results showed remarkable consistency, leading us to adopt the final configuration of $(0.10, 0.85, 0.05)$. This configuration reveals that myopically pursuing completion time is detrimental, while minimizing wasted space is the dominant factor for high-quality solutions. When the algorithm processes the final segment of tasks, it dynamically switches its objective to solely minimizing the completion time.

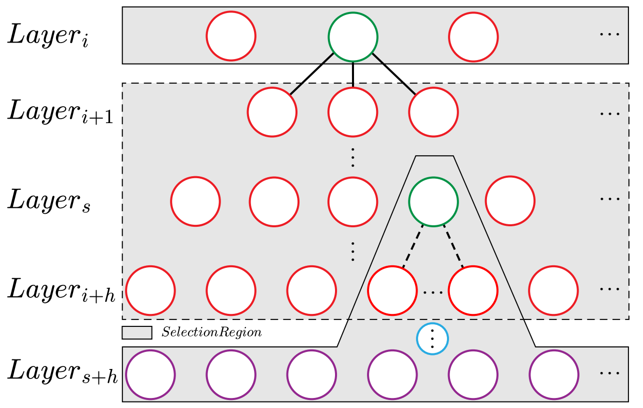


Fig. 8: The state transition and selection graph of S-HDP. The figure illustrates two consecutive selection steps. In the first step (green and red states), the algorithm explores all layers in a lookahead "Selection Region" of height h. A holistic scoring function selects the best state (green). The search then advances from this chosen state, initiating a new lookahead "Selection Region" (blue and purple states), from which the next state is selected in the same manner. This comprehensive, multi-step evaluation avoids the shortsightedness of simpler S-G and S-SDP.

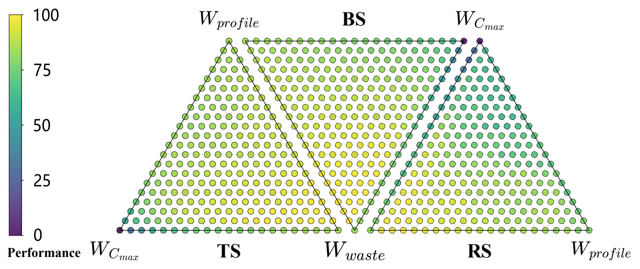
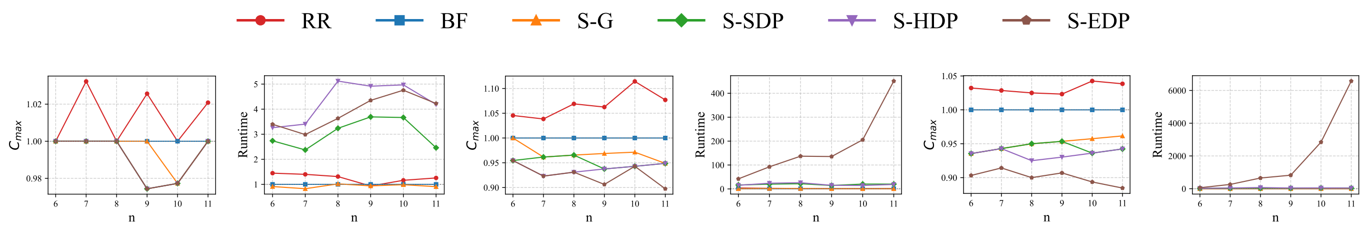


Fig. 9: Tuning the weights of the Heuristic DP scoring function via simulation. Each ternary plot shows the solution quality (color-coded **Performance**) across different weightings of the three metrics: completion time ($W_{\Delta C_{max}}$), waste rate (W_{waste}), and profile quality ($W_{profile}$). Here, **Performance** indicates the average C_{max} across all experiments and problem scales for given weights. The experiment is performed on three distinct scenarios to ensure robustness: TS, BS, and RS. Across all scenarios, the highest-performing configurations (yellow areas) consistently prioritize minimizing the waste rate, revealing it as the most critical factor for achieving a high-quality schedule.

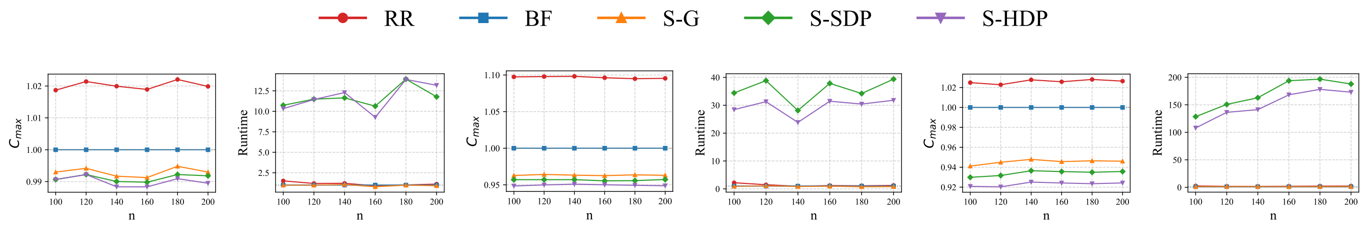
VII. Simulations

The preceding sections established a theoretical framework for the LSR-MUCSP and designed a series of scheduling algorithms. This section aims to comprehensively evaluate the practical performance of these algorithms through computational simulations. The simulations compare our S-EDP, S-SDP, S-G and S-HDP against two baseline algorithms, examining both solution quality (completion time) and computational efficiency (runtime).



(a) RS C_{max} (b) RS Runtime (c) BS C_{max} (d) BS Runtime (e) TS C_{max} (f) TS Runtime

Fig. 10: Performance comparison in small-scale scenarios ($n \leq 11$). The y-axis represents the average completion time ($C_{\max}$) and runtime, with all values normalized by the results of BF.



(a) RS C_{max} (b) RS Runtime (c) BS C_{max} (d) BS Runtime (e) TS C_{max} (f) TS Runtime

Fig. 11: Performance comparison in large-scale scenarios ($n \geq 100$). The y-axis represents the average completion time ($C_{\max}$) and runtime, normalized by BF. S-EDP is excluded due to its computational infeasibility at this scale.

A. Simulation Setup

All algorithms were implemented in Rust and executed on a computer with an AMD Ryzen 9 7945HX CPU and 16GB of RAM. The evaluation is divided into small-scale and large-scale scenarios. We compare our proposed algorithms (*S-EDP*, *S-SDP* ($a = 5$), *S-G*, and *S-HDP* ($a_{max} = 5$, $W = (0.10, 0.85, 0.05)$)) against two baseline methods that do not use the skyline framework:

- Round-Robin *RR* [24]: A classic scheduling algorithm representing a **load-balancing** approach. This algorithm implements a cyclic scheduling policy for fair workload distribution. It assigns task t_i to a contiguous block of m_i UAVs selected by a sequential pointer, setting the start time to the maximum availability among them.
- Best-Fit *BF* [20], [21], [17]: Inspired by the two-dimensional strip packing and the game-theoretic auction mechanism known as the **Contract Net Protocol (CNP)**, BF is a classic and adaptive greedy heuristic. Its logic is equivalent to an **auction-based game-theoretic mechanism**. In this model, the task t_i acts as the "auctioneer" seeking to be scheduled. The $M - m_i + 1$ contiguous UAV blocks act as "bidders," each submitting a "bid" corresponding to the earliest start time it can offer. The task t_i is then awarded to the bidder that submitted the winning bid (i.e., earliest start time).

The performance of these algorithms is evaluated across three distinct scenarios designed to test their robustness: TS, BS, and RS. The detailed characteristics and parameter generation for these scenarios are identical to those described in Subsection VI-C.

B. Results and Analysis

Our simulation results, presented for small-scale ($n \leq 11$) in Fig. 10 and large-scale ($n \geq 100$) problems in Fig. 11, validate our proposed framework. The primary and most consistent finding is that all skyline-based algorithms (S-G, S-SDP, and S-HDP) demonstrate a significant and robust performance advantage in solution quality ($C_{\max}$) over the conventional RR and BF baselines across all scenarios. This fundamental advantage is starkly highlighted by the fact that even the simplest online greedy implementation, S-G, yields better solutions than the adaptive BF heuristic, underscoring the power of the skyline representation itself.

Small-Scale Scenarios. In tests where the true optimum is known via S-EDP, the performance hierarchy within the skyline framework and the near-optimality of S-HDP are evident. As heuristic sophistication increases from S-G to S-SDP and finally to S-HDP, the solution quality consistently improves. S-HDP distinguishes itself by achieving an average completion time that deviates less than 1% from the optimal results provided by S-EDP. The runtime analysis confirms S-EDP's exponential complexity, motivating the need for heuristics. In contrast, all heuristics exhibit excellent efficiency, with the substantial gains in solution quality achieved.

Large-Scale Scenarios. The performance hierarchy is robust and holds true for large-scale problems where S-EDP is computationally infeasible. S-HDP continues to deliver the best solutions, establishing it as the most effective algorithm for practical applications. For instance, in the TS scenario with $n = 120$, S-HDP improves upon the completion time of the RR baseline by approximately 10%. The runtimes of all heuristics exhibit manageable, polynomial-level growth, confirming their suitability for real-world demands.

The performance gaps between algorithms are heavily influenced by the task profile. The differences are most pronounced in the Time-dominant (TS) scenario and narrowest in the Resource-dominant (RS) scenario. This is because the high, concurrent resource demands in the RS scenario create a natural bottleneck that forces a near-sequential execution with limited optimization potential for any algorithm. Conversely, the TS scenario, with its long-duration but low-resource tasks, presents a more complex combinatorial puzzle that amplifies the performance differences between superior and conventional scheduling strategies.

VIII. Conclusion

In this paper, we formalized the Single-Route Multi-UAV Cooperative Scheduling Problem (SR-MUCSP), a computationally challenging problem motivated by emerging applications in the low-altitude economy. To tackle this, we propose a novel skyline-based scheduling framework, using a "skyline" vector to create a canonical representation of resource availability over time. This core concept enabled the design of a provably optimal exact algorithm, S-EDP, which is made tractable for small-scale instances by two powerful, problem-specific pruning strategies: state dominance and Candidate Placement Points (CPPs).

Our comprehensive simulations validated the effectiveness of this framework. The results demonstrated that all algorithms derived from the skyline representation consistently and significantly outperform conventional heuristics across a variety of task profiles. Furthermore, to handle large-scale instances, we developed a suite of high-performance heuristics. Notably, our premier heuristic, S-HDP, successfully balances near-optimal solution quality with high computational efficiency, establishing it as a robust and practical solution for real-world applications.

This work also opens avenues for future research. Our current model assumes a homogeneous UAV fleet; a key future direction is to accommodate heterogeneous UAVs with varying capabilities, which presents a significant challenge.

References

[1] X. Liu, "An overview of low-altitude economy research: Evolutionary trajectory, core controversies and future pathways," *Journal of Economic Policy and Finance*, vol. 11, no. 3, pp. 187–193, 2025.

[2] DJI, "DJI Dock 3 - Rise to Any Challenge," https://enterprise.dji.com/cn/dock-3, 2025, accessed: 2025-08-27.

[3] H. Ghazzai, H. Menouar, A. Kadri, and Y. Massoud, "Future uav-based its: A comprehensive scheduling framework," *IEEE Access*, vol. 7, pp. 75 678–75 695, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:195222532

[4] Y. Qu, H. Sun, C. Dong, J. Kang, H. Dai, Q. Wu, and S. Guo, "Elastic collaborative edge intelligence for uav swarm: Architecture, challenges, and opportunities," *IEEE Communications Magazine*, vol. 62, pp. 62–68, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:264459895

[5] N. Mohamed, J. Al-Jaroodi, I. Jawhar, A. Idries, and F. Mohammed, "Unmanned aerial vehicles applications in future smart cities," *Technological Forecasting and Social Change*, vol. 153, p. 119293, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0040162517314968

[6] W. Y. H. Adoni, S. Lorenz, J. S. Fareedh, R. Gloaguen, and M. Bussmann, "Investigation of autonomous multi-uav systems for target detection in distributed environment: Current developments and open challenges," *Drones*, vol. 7, no. 4, 2023. [Online]. Available: https://www.mdpi.com/2504-446X/7/4/263

[7] Y. Qu, H. Dai, H. Wang, C. Dong, F. Wu, S. Guo, and Q. hui Wu, "Service provisioning for uav-enabled mobile edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 39, pp. 3287–3305, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:237804523

[8] Y. Qu, H. Dai, L. Wang, W. Wang, F. Wu, H. Tan, S. Tang, and C. Dong, "Cotask: Correlation-aware task offloading in edge computing," *World Wide Web*, vol. 25, pp. 2185 – 2213, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:249803439

[9] F. Shan, J. Huang, R. Xiong, F. Dong, J. Luo, and S. Wang, "Energy-efficient general poi-visiting by uav with a practical flight energy model," *IEEE Transactions on Mobile Computing*, vol. 22, no. 11, pp. 6427–6444, 2022.

[10] Y. Wang, J. Huang, F. Shan, Y. Gao, R. Xiong, and J. Luo, "Optimizing joint speed and altitude schedule for uav data collection in low-altitude airspace," *IEEE Transactions on Mobile Computing*, pp. 1–15, 2025.

[11] H. Dai, K. Sun, A. X. Liu, L. Zhang, J. Zheng, and G. Chen, "Charging task scheduling for directional wireless charger networks," *IEEE Transactions on Mobile Computing*, vol. 20, no. 11, pp. 3163–3180, 2021.

[12] Y. Shoham and K. Leyton-Brown, *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2009.

[13] D. P. Bertsekas, "The auction algorithm: A distributed relaxation method for the assignment problem," *Annals of Operations Research*, vol. 14, no. 1, pp. 105–123, 1988.

[14] O. Shehory and S. Kraus, "Task allocation via coalition formation among autonomous agents," in *International Joint Conference on Artificial Intelligence*, 1995. [Online]. Available: https://api.semanticscholar.org/CorpusID:2217087

[15] Y. Li, Z. Zhang, Z. He, and Q. Sun, "A heuristic task allocation method based on overlapping coalition formation game for heterogeneous uavs," *IEEE Internet of Things Journal*, vol. 11, pp. 28 945–28 959, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:270089279

[16] T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohmé, "Coalition structure generation with worst case guarantees," *Artificial Intelligence*, vol. 111, no. 1-2, pp. 209–238, 1999.

[17] R. G. Smith, "The contract net protocol: High-level communication and control in a distributed problem solver," *IEEE Transactions on Computers*, vol. C-29, pp. 1104–1113, 1980. [Online]. Available: https://api.semanticscholar.org/CorpusID:15267324

[18] F. Fioretto, E. Pontelli, and W. G. S. Yeoh, "Distributed constraint optimization problems and applications: A survey," *ArXiv*, vol. abs/1602.06347, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:4503761

[19] H. Luan, Y. Xu, D. Liu, Z. Du, H. Qian, X. Liu, and X. Tong, "Energy efficient task cooperation for multi-uav networks: A coalition formation game approach," *IEEE Access*, vol. 8, pp. 149 372–149 384, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:221282150

[20] A. Lodi, S. Martello, and M. Monaci, "Two-dimensional packing problems: A survey," *Eur. J. Oper. Res.*, vol. 141, pp. 241–252, 2002. [Online]. Available: https://api.semanticscholar.org/CorpusID:3240079

[21] D. S. Johnson, A. J. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham, "Worst-case performance bounds for simple one-dimensional packing algorithms," *SIAM J. Comput.*, vol. 3, pp. 299–325, 1974. [Online]. Available: https://api.semanticscholar.org/CorpusID:14214580

[22] J. Martinovic and N. Strasdat, "Theoretical insights and a new class of valid inequalities for the temporal bin packing problem with fire-ups," *Pesquisa Operacional*, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:249009455

[23] J. H. van Lint and R. M. Wilson, *A Course in Combinatorics*, 2nd ed. Cambridge University Press, 2001.

[24] Z. Feng, W. Xu, and J. Cao, "Distributed nash equilibrium computation under round-robin scheduling protocol," *IEEE Transactions on Automatic Control*, vol. 69, pp. 339–346, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:257829320