Hitchhiker's Guide to Patrolling: Path-Finding for Energy-Sharing Drone-UGV Teams

Jonathan Diller
Colorado School of Mines
Golden, United States
jdiller@mines.edu

Qi Han
Colorado School of Mines
Golden, United States
qhan@mines.edu

Robert Byers
Colorado School of Mines
Golden, United States
robert_byers@mines.edu

James Dotterweich
DEVCOM Army Research Laboratory
Aberdeen, United States
james.m.dotterweich.civ@army.mil

James Humann
DEVCOM Army Research Laboratory
Aberdeen, United States
james.d.humann.civ@army.mil

ABSTRACT

Teams of Unmanned Ground Vehicles (UGVs) and drones are often proposed for various patrolling applications, where drones quickly move from one point of interest to the next while UGVs act as moving base stations that can both recharge and ferry around the drones. In this paper, we look at how to plan collaborative actions between drones and UGVs for a patrolling mission over an indefinite time horizon. We demonstrate how to form a second-order cone (SOC) program that finds optimal solutions, in polynomial time, to a variant of the larger problem where the order of drone and UGV actions are fixed. We propose two algorithms that use our SOC program to find locally optimal solutions while considering the limited energy of both UGVs and drones. Our numerical simulation results show that both of our algorithms yield a greater than 50% improvement in solution quality when compared to a baseline method from the literature. Additionally, we demonstrate the authenticity of our problem setup through a proof-of-concept experiment on a physical UGV and drone testbed.

KEYWORDS

Multi-Agent Planning; Energy-Aware Planning; Resource Sharing

ACM Reference Format:
Jonathan Diller, Qi Han, Robert Byers, James Dotterweich, and James Humann. 2025. Hitchhiker's Guide to Patrolling: Path-Finding for Energy-Sharing Drone-UGV Teams. In *Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Detroit, Michigan, USA, May 19 – 23, 2025*, IFAAMAS, 9 pages.

1 INTRODUCTION

Unmanned aerial vehicles (UAVs - commonly referred to as drones) are fast and agile but have limited energy. Alternatively, unmanned ground vehicles (UGVs) have larger battery storage and can be used to recharge drones but are slower and may struggle to traverse rough terrain. These two platforms, when paired together, can serve as an autonomous monitoring and patrolling team where the drone is able to quickly move from one Point of Interest (PoI) to the next while the UGV acts as a moving base station that can both recharge and ferry around hitchhiking drones.

Path finding and rendezvous planning for drone-UGV teams is a well studied area of interest within the robotics community. A large body of this research focuses on minimizing the maximum latency that a PoI is visited, often referred to minimizing the "worst latency". However, we argue that prioritizing the PoI with the worst latency does not accurately represent how well a team is patrolling a set of PoIs and may lead to other PoIs being ignored. Consider a scenario where robots must routinely inspect equipment at a petroleum refinery for hazardous air pollutant leaks [1, 29]. Ignoring a leak for too long will not only lead to a larger pollutant spill, but may cause the leak to become worse over time. Therefore, the penalty for neglecting any individual inspection site should grow exponentially with time. In applications where a penalty can be assigned for neglecting any individual PoI, such as security, maintenance inspections, or sanitation, we argue that minimizing the rate the penalty is accrued over an indefinite time horizon should be preferred over minimizing the maximum latency.

Additionally, previous works have failed to consider how to plan for long-term or indefinite deployments taking into account energy requirements of these platforms. Moreover, many existing works on this subject tend to simplify models for energy consumption and energy transfer from the UGV to the drone, or ignore energy constraints all together. This article seeks to fill this gap in the existing literature.

In this work, we address the problem of planning patrolling routes over indefinite time horizons for energy-constrained drone and UGV teams, where UGVs can transfer energy to and ferry around hitchhiking drones. Our unique contributions include a Second-Order Cone program for finding local optimum in polynomial time for drone-UGV team monitoring problems, a framework for solving drone-UGV team monitoring problems that considers energy sharing and UGV energy constraints, and a taxonomy of drone-UGV team monitoring problems.

2 LITERATURE REVIEW

In this section we define a taxonomy for Drone-UGV monitoring problems and then summarize algorithms for these problems.

Distribution Statement A. Approved for public release: distribution is unlimited.
This work was supported in part by the Journeyman Fellowship (W911NF-24-2-0140).

Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Y. Vorobeychik, S. Das, A. Nowé (eds.), May 19 – 23, 2025, Detroit, Michigan, USA. © 2025 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

2.1 Related Drone-UGV Problems

In this article, we look at a Drone-UGV teams monitoring problem where mixed drone-UGV teams must visit a set of PoIs. The specific application may vary widely, such as agriculture monitoring, border patrolling, sanitation, environmental monitoring, maintenance inspections, or perimeter patrolling, and may involve different types of tasks at each PoI, such as taking images, taking a sensor measurement, or collecting data from wireless nodes, but the fundamental planning problem remains the same: find paths that visit each PoI while optimizing some metric.

Variations of the PoI visiting problem can be classified based on a variety of factors. These factors include:

(1) the number of times a PoI is visited (fixed visit requirement – **F**, or repeated visits indefinitely – **I**),
(2) the considered metric (minimizing completion time – **T**, minimizing maximum latency – **L**, minimizing distance – **D**, or some other metric – **O**),
(3) which vehicle can visit a PoI (drone only – **C**, or drone and UGV – **B**), and
(4) what support the ground vehicle provides to the drone (energy transfer only – **E**, position transfer only – **P**, or both energy and position transfer – **H** – a.k.a hitchhiking).

Note that the difference between the T and L metrics is that minimizing completion time (T) considers the time required to visit *all* PoIs while minimizing maximum latency (L) considers the time to visit the single most neglected PoI. Metric T tends to occur when there is only a single robot (e.g. a single drone) that is visiting PoIs, while metric L tends to come up when there are more than one robot visiting PoIs in parallel (e.g. multiple drones), although these trends are not strictly followed. Examples of other, less common metrics (O) include time penalties [18] and average latency [30].

The majority of existing work falls within F:T variations of the problem. Works looking at minimizing completion time for fixed visits where only drones can visit PoIs (F:T:C) include [7, 14, 31, 32]. The natural extension to this scenario setup is to allow UGVs to also visit PoIs (F:T:B), as seen in [11, 19, 21]. An example where distance is used as a metric instead of time (the F:D:C variation) can be found in [22].

Work on indefinite PoI monitoring with drone-UGV teams, often termed patrolling, is more sparse. Examples where drones are responsible for visiting PoIs while minimizing completion time (I:T:C) are found in [16, 17] while minimizing maximum latency examples (I:L:C) are seen in [13, 25]. In [18], they consider a patrolling problem where both the drone and UGV can visit PoIs while using a penalty that is measured at the end of a fixed time period (I:O:B). There are also many examples of minimizing maximum latency in patrolling problems outside of drone-UGV teams, such as [6, 23, 24], which fall into the generalized multi-robot patrolling problem [3].

How UGVs are used in path finding problems vary across both fixed visit and repeated visits scenarios. Works that only use the UGV for energy transfer (E) include [11, 17–19, 21, 22, 31, 32]. The more complex hitchhiking setup, where the UGV can both recharge and ferry about drones, is less common but seen in [7, 13, 14, 25].

In this work, we consider a variation of the Drone-UGV teams PoI monitoring problem where PoIs are visited indefinitely by drones that hitchhike off of the UGV while minimizing the rate that a penalty is accumulated for neglecting PoIs (I:O:C:H). To the best of our knowledge, we are the first to consider this variation of the Drone-UGV teams PoI monitoring problem. Perhaps the most similar work to ours is found in [13, 25], where they have a similar problem setup but are minimizing the maximum latency (I:L:C:H). Although minimizing maximum latency may be preferable for certain applications, we argue that this metric is less desirable in applications where a penalty can be assigned for neglecting any individual PoI. We prove that minimizing the maximum latency does not decrease the penalty accumulation rate and use the algorithm proposed in [25] as our baseline in this paper.

Furthermore, none of the summarized existing literature considers limited onboard energy for the UGV. This is particularly problematic for indefinite patrolling problems, where the UGV will eventually run out of energy. Additionally, all existing works assume that energy transfer from the UGV to the drone is completed at a fixed time interval. Although this assumption may be reasonable for scenarios where "energy transfer" refers to swapping batteries, the time required to recharge drone batteries will depend on how much energy was consumed by the drone prior to recharging. In this work, we consider the limited energy of both drones and UGVs and incorporate a model for wirelessly recharging drones on a UGV.

2.2 Path-Finding Algorithms

There are many approaches to solving the different variations of the Drone-UGV teams PoI monitoring problem. The most common approach is to first find routes for UGVs then build drone routes off of static UGV routes, as seen in [7, 13, 21, 22, 25], although some have considered UGV and drone routing as separate problems and focus on finding rendezvous points [2, 11, 32]. An example of planning routes together is found in [31].

Regardless of how these problems are broken up, these algorithms usually run iteratively [7, 19, 21], where different sub-problems are addressed individually on each iteration. The iterative approach commonly uses Integer Programming [7, 16, 19, 31], though Integer Programs do not run in polynomial time. A common method for separating the work done by different robots is to use a cluster algorithm, as seen in [14, 22] or other partitioning algorithms [13, 25]. Finding routes themselves is often treated as a Traveling Salesman Problem, for which there are well studied polynomial time solutions and complete solutions [8].

In this work, we propose a Second-Order Cone (SOC) program to optimize routes for both drones and UGVs together. The advantage to formulating this problem as a SOC program is that they can be solved to optimality in polynomial time. Both of our proposed algorithms to the Team-PITH problem utilize this SOC program. The second algorithm runs iteratively, solving the SOC program and the TSP repeatedly, while the first algorithm shows the effectiveness of the SOC program without iterations.

3 PROBLEM FORMULATION

We consider a problem where collaborative teams of UGVs and drones must repeatedly visit PoIs with the objective of reducing the time that any PoI is left unattended. For this problem, we are given m_g UGVs, m_a drones, and n PoIs that must be routinely visited by one (or more) of the drones. Let I be the set of all PoIs, R^a be the

set of drones, and R^g be the set of UGVs, where $|I| = n$, $|R^a| = m_a$, and $|R^g| = m_g$. Each UGV has an assigned team of drones that can land and launch off of the UGV and recharge from the UGV. Let D_j be the set of drones assigned to UGV j. We assume that all D_j are fixed and leave mixing UGV-drone teams to future work.

In the following subsections we mathematically define a patrolling kernel and the penalty accrued for leaving a PoI unmonitored, discuss energy models for both drones and UGVs, including wireless energy transfer from a UGV to a drone, and formally define the Team-PITH Problem.

3.1 Patrolling Kernels

To define a patrolling route for each robot, we use sequences of actions that dictate what each vehicle does. We define action i as the tuple $a_i = \{(x_i, y_i), t_i, \varsigma\}$, where:

- (x_i, y_i) are the x, y-coordinates where the action takes place,
- t_i is the time that the action is completed, and
- ς represents the actions type, defined over a set of possible action types for each robot type.

For drones, action types include *take-off*, *visit-PoI*, *return-to-UGV* and *land*. For UGVs, action types include *launch-drone*, *receive-drone*, *return-to-base* and *swap-battery*. Let A^g_j be a sequence of actions for UGV j and A^a_k be a sequence of actions for drone k. Note that consecutive pairs of actions in an action sequence may require the robot to move from one location to the next and perform some procedure at that location. For example, a *launch-drone* action a_i at some (x_i, y_i) position followed by a *receive-drone* action a_{i+1} at a different (x_{i+1}, y_{i+1}) position will require the UGV to physically move from (x_i, y_i) to (x_{i+1}, y_{i+1}) and wait for the drone to land before action a_{i+1} can be considered complete. We say that an action sequence, A^g_j or A^a_k, is *valid* if every consecutive pair of actions in the action sequence adhere to the dynamic limitations of the robot and allow time for the actions type to be completed.

Each time a drone performs a *take-off* action followed by some number of *visit-PoI* actions, a *return-to-UGV* action, and a *land* action, it is termed a sortie (that is, a complete drone flight). We formally define a drone sortie as the tuple $\mathcal{T}_k = \{{}^s a_k, {}^e a_k, I_k\}$, where

- ${}^s a_k$ is the *take-off* action that starts the sortie,
- ${}^e a_k$ is the *land* action that ends the sortie, and
- I_k is a sequence of PoIs that the drone will visit along the sortie.

Each A^a_k can contain multiple $\mathcal{T}_k$.

We define a patrolling kernel as the set $\mathcal{S} = \{A^g, A^a\}$, where

- A^g is a set of UGV action sequences, and
- A^a is a set of drone action sequences.

Observe that certain drone and UGV actions will be correlated. For example, if drone k is launching from UGV j, then there should be a *take-off* action $a_k \in A^a_k$ that corresponds with some *launch-drone* action $a_j \in A^g_j$ where $t_k = t_j$ for $t_k \in a_k$, $t_j \in a_j$. We say that $\mathcal{S}$ is *consistent* if *take-off* and *land* actions in A^a have a corresponding *launch-drone* and *receive-drone* action in A^g.

3.2 Penalty Accumulation Rate

In this work, we propose imposing a penalty for neglecting a PoI and increasing how much penalty is applied as time passes. The idea is to prevent the robots from ignoring a PoI for too long while also factoring in to the objective how often *all* PoIs are monitored.

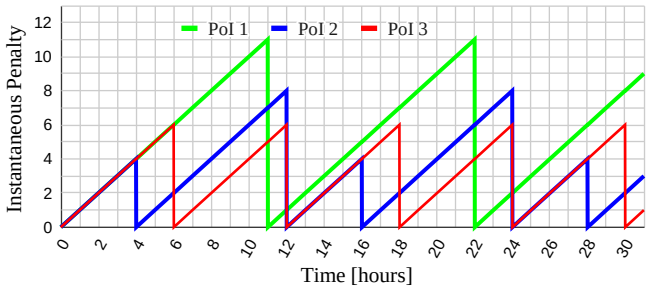

Figure 1: Examples of Instantaneous Penalty for PoIs.

Let the instantaneous penalty be linearly proportional to the units of time of the PoI latency. That is, for each hour that passes that we do not remove trash from a trash bin, the rate that we gain a penalty grows by one unit. The instantaneous penalty resets to zero each time a PoI is visited. An example of this is shown in Figure 1, where three PoIs are visited over varying intervals and the instantaneous penalty grows linearly with time. Suppose that a robot removed the trash every t_i hours. The total penalty that we gain over t_i hours will be $\int_0^{t_i} t\, dt = \frac{1}{2} t_i^2$. The total penalty gained at any given time is the area under the latency lines, which is the black line in Figure 2.

We chose hours as our unit of time because it lends itself well to something easily interpretable by humans (removing trash every 5 hours is more tangible than saying every 18,000 seconds), but one could switch to any other unit of time (seconds, days, galactic-years) as is appropriate for the application and the following math would still apply.

Given patrolling kernel $\mathcal{S}$, we define a PoI visit sequence as $S^v_i = \{t^i_1, t^i_2, \cdots, t^i_L\}$ for each PoI $i \in I$, where each t^i_l is the time lapsed in hours from the beginning of $\mathcal{S}$ until i is visited by a drone in an action sequence in $\mathcal{S}$ for the l^{th} time, for $l \in \{1, 2, \cdots, L\}$. We assume that the patrolling kernel $\mathcal{S}$, and all S^v_i, will be repeated indefinitely. From visit sequence S^v_i, we can derive the latency sequence of kernel $\mathcal{S}$ as $S^L_i = \{t^i_1, (t^i_2 - t^i_1), \cdots, (t^i_L - t^i_{L-1})\}$, which gives us the latency intervals that i is visited in. From Figure 1, $S^L_1 = \{11\}$, $S^L_2 = \{4, 8\}$ and $S^L_3 = \{6\}$.

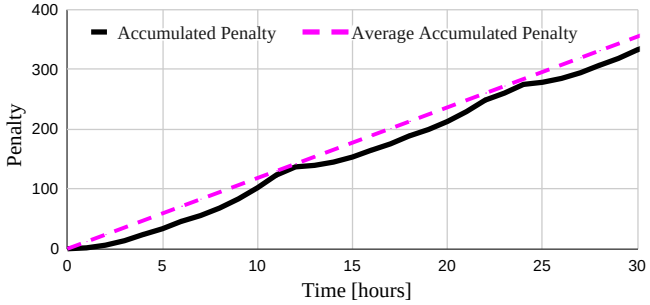

Figure 2: Point of Interest penalty accumulation example.

The penalty accumulated for a single PoI by the end of a latency sequence will be

$$F_p(S_i^L) = \sum_{t_l \in S_i^L} \frac{(t_l)^2}{2}. \quad (1)$$

The average rate that penalty across all PoIs is accrued as patrolling continues indefinitely will then be

$$PAR(\mathcal{S}) = \sum_{i \in I} \frac{1}{t_L^i} F_p(S_i^L). \quad (2)$$

We term (2) the Penalty Accumulation Rate (PAR). From Figure 1, $PAR(\mathcal{S}) = (\frac{11^2}{2})(\frac{1}{11}) + (\frac{4^2+8^2}{2})(\frac{1}{12}) + (\frac{6^2}{2})(\frac{1}{6}) = 11.833$, which is the slope of the pink dashed line in Figure 2.

Using PAR as a metric gives us a way to check how well a given $\mathcal{S}$ addresses all PoIs, as oppose to using minimizing the maximum latency, which does not tell us how well a patrolling kernel is monitoring the non-maximum latency PoIs. In fact, we can make a strong distinction between these two metrics:

Theorem 1. *Minimizing the maximum latency across all PoIs does not always reduce PAR.*

Proof. By example, suppose that we found a new patrolling kernel $\mathcal{S}'$ with latency sequences $S_1^L = \{10\}$, $S_2^L = \{10\}$ and $S_3^L = \{8\}$. Kernel $\mathcal{S}'$ has a maximum latency of 10 while $\mathcal{S}$ has a maximum latency of 11. However, $PAR(\mathcal{S}') = 14 \not< PAR(\mathcal{S}) = 11.833$. □

3.3 Robot Energy Models

To formulate our patrolling problem, we require accurate models for UGV energy consumption, drone energy consumption, and wireless energy transfer from a UGV to a drone.

Energy models for UGVs: Previous works have shown that the rate that a UGV consumes energy (in watts) depends on the speed of the vehicle and can broadly be modeled as:

$$\mathcal{P}_g(v) = c_1 v + c_2 \quad (3)$$

where v is the speed that the vehicle is moving at (in meters per second), and c_1 and c_2 are constants. An example of this model is found in [10], where they experimentally found $c_1 = 464.8$ and $c_2 = 156.3$ for a standard Clearpath™ Warthog UGV.

Energy models for drones: Similarly, works such as [15, 33] have shown that the power consumption on drones also depends on the speed of the drone and can generally be approximated as:

$$\mathcal{P}_a(v) = c_3 v^3 + c_4 v^2 + c_5 v + c_6 \quad (4)$$

where c_3, c_4, c_5 and c_6 are again constants. An example of this model is found in [26] where they experimentally determined these values for a custom build hexacopter. We experimentally found $c_3 = 0, c_4 = 0, c_5 = -1.695$ and $c_6 = 396.74$ on the custom built drone found in Section 6. In addition to Equation (3), we assume that launching and receiving a drone will consume a constant amount of energy. Let ${}^{l}J_k$ and ${}^{r}J_k$ be the energy required to launch and receive drone j, respectively.

Wireless energy transfer from UGVs to drones: Unmanned ground vehicles can hold significantly larger volumes of usable energy when compared to drones, making them ideal for sharing stored energy with their energy-limited drone teammates. To charge a drone's battery, we charge at constant current up to a "crossover point", E^* (in joules), where energy transfer slows down. After reaching E^* jules, we apply a constant voltage until the battery is full. Therefore, the relationship between energy transferred per unit of time can be modeled as:

$$E_t(t) = \begin{cases} c_7 t^2 + c_8 t & 0 \le t \le t^* \\ E^* + \frac{P^*}{\alpha}(1 - e^{-\alpha(t-t^*)}) & t^* \le t \le t_{max} \end{cases} \quad (5)$$

where t is the charge time in seconds, t^* is the time required to reach the crossover point E^*, P^* is the power level at E^*, and α, c_7 and c_8 are all constants determined by battery and hardware characteristics.

From Equation (5), we find that the time required to recharge a drone after consuming E joules will be:

$$t_c(E) = \begin{cases} \frac{-c_8 + \sqrt{c_8^2 + 4c_7 E}}{2c_7} & E_{min} \le E < E^* \\ t^* - \frac{ln\left(1 + \frac{\alpha}{P^*}(E^* - E)\right)}{\alpha} & E^* \le E \le E_{max} \end{cases} \quad (6)$$

where E_{min} is the minimum allowable energy level that the battery can safely be discharged to and E_{max} is the capacity of the battery. We expect to lose γ percent of energy in the wireless transfer process.

Applying Energy Models: As an example on how to apply these equations, suppose that drone k performs sortie $\mathcal{T}_k$ and moves at speed v_k. The drone will consume

$$\mathbf{J}_a(\mathcal{T}_k) = {}^{l}J_k + \frac{d_k}{v_k}\mathcal{P}_a(v_k) + {}^{r}J_k \quad (7)$$

joules of energy, where d_k is the total distance required to travel from the *take-off* action to each PoI in I_k then return to the *land* action in $\mathcal{T}_k$. It will then take $t_c(\mathbf{J}_a(\mathcal{T}_k))$ seconds to recharge the drone on the UGV and the UGV will lose $(1+\gamma)\mathbf{J}_a(\mathcal{T}_k)$ joules from its own battery. We say that drone action sequence A_k^a is *safe* if for all consecutive sorties $\mathcal{T}_k, \mathcal{T}_{k+1} \in A_k^a$, sortie $\mathcal{T}_{k+1}$ does not begin until $t_c(\mathbf{J}_a(\mathcal{T}_k))$ seconds after sortie $\mathcal{T}_k$ ends. That is, A_k^a is *safe* if we fully recharge the drone before launching it again.

Let ${}^{max}J_k$ be the maximum usable energy stored on drone k. Each second that the drone is operating it will consume $\mathcal{P}_a(v_k)$ joules per second and can operate for

$${}^{max}t_k = \frac{{}^{max}J_k - ({}^{l}J_k + {}^{r}J_k)}{\mathcal{P}_a(v_k)} \quad (8)$$

seconds and travel ${}^{max}d_k$ meters. If the drone consumes all usable energy on-board, it will take ${}^{max\text{-}charge}t_k = t_c({}^{max}J_k)$ seconds to recharge the battery completely.

In a similar manner, the energy consumed by UGV j after completing action sequence A_j^g (excluding energy transferred to drones), will be

$$\mathbf{J}_g(A_j^g) = \frac{d_j}{v_j}\mathcal{P}_g(v_j) \quad (9)$$

where d_j is the total travel distance to complete the actions in A_j^g and v_j is the speed of the UGV. Let ${}^{max}J_j$ be the maximum usable energy stored on UGV j.

We acknowledge that these energy models abstract other factors in energy consumption, such as environmental conditions or payload [34]. However, we argue that these aspects could be included into the models without changing the fundamental approach.

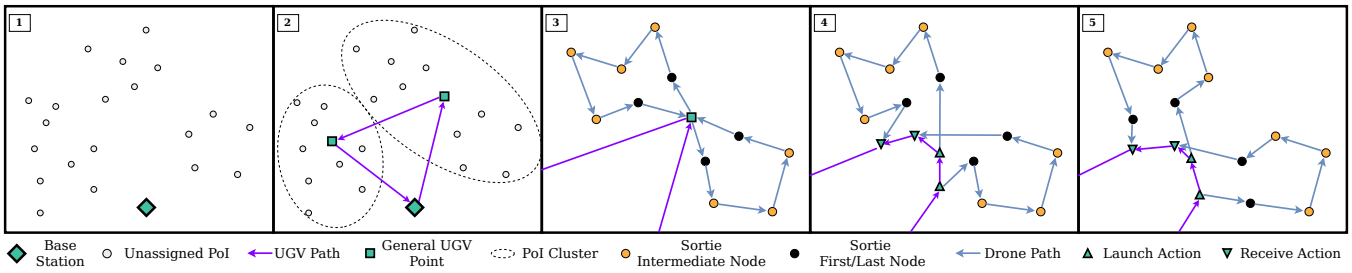

Figure 3: Example of the ILO algorithm running on a simple input.

3.4 Formal Problem Definition

Given m_g UGVs, m_a drones, and n PoIs, find a *consistent* patrolling kernel $\mathcal{S}$ with *valid* robot action sequences such that $PAR(\mathcal{S})$ is minimized, subject to:

$$\mathcal{S} \text{ is } \textit{consistent} \tag{10a}$$

$$\mathcal{S} \text{ is } \textit{valid} \tag{10b}$$

$$\mathbf{J}_g(A_j^g) + \sum_{k \in D_j} (1+\gamma)\mathbf{J}_a(A_k^a) \leq {}^{max}J_j, \qquad \forall A_j^g \in \mathcal{S} \tag{10c}$$

$$\mathbf{J}_a(A_k^a) \leq {}^{max}J_k, \qquad \forall A_k^a \in \mathcal{S} \tag{10d}$$

$$A_k^a \text{ is } \textit{safe}, \qquad \forall A_k^a \in \mathcal{S} \tag{10e}$$

We term this the Team Patrolling over Indefinite Time Horizons (Team-PITH) Problem.

We can show that Team-PITH is $\mathcal{NP}$-Hard through a reduction from the Traveling Salesman Problem (TSP), a well-known $\mathcal{NP}$-Hard problem [12].

Theorem 2. *The Team-PITH problem is $\mathcal{NP}$-Hard.*

Proof. Given graph $G = \{V, E\}$ as an input to the TSP, where V is a set of vertices and $E = V \times V$ is a set of edges, we can reduce the TSP to the Team-PITH problem via the following procedure: Set $I = V$, place the base station at the center of V, and $m_g = m_a = 1$ (a single UGV and a single drone). Set ${}^{max}J_k$, the usable energy onboard the drone, to an arbitrarily high value and set the UGV speed to 0. After solving the Team-PITH problem, $A^a \in \mathcal{S}$ will contain a closed cycle over the points in I, which provides a solution to the original TSP. Because TSP is $\mathcal{NP}$-Hard, and we provided a polynomial time reduction from TSP to the Team-PITH problem, the Team-PITH problem must also be $\mathcal{NP}$-Hard. □

4 PROPOSED ALGORITHMS

In this section we introduce the Launch Optimizer (LO) and Iterative Launch Optimizer (ILO) algorithms for the Team-PITH problem and discuss their individual parts in detail in the subsequent subsections. We also provide a time complexity analysis on these algorithms.

4.1 Launch Optimizer Algorithms

We will first describe the ILO algorithm then discuss how the more simplistic LO algorithm differs from this first algorithm. In general, the ILO algorithm takes an initial solution (Figure 3, panels 1-3), optimizes drone launch and land locations across each UGV tour using a SOC program, and then attempts to further improve the solution by iteratively updating drone sorties and solving the SOC program again (Figure 3, panel 5).

Listing 1 lays out the details of the ILO algorithm. The algorithm finds an initial solution, $\mathcal{S}$, using the *Initial-Solution()* function on line 1. The details of the *Initial-Solution()* function are further discussed in Section 4.2. After finding an initial solution, we create a temporary new solution $\mathcal{S}'$ that acts as our interim solution for algorithm steps 2 through 13. For each UGV $j \in R^g$ and each drone $k \in R^a$ assigned to UGV j, we update interim solution $\mathcal{S}'$ by attempting to improve the ordering of PoI visit actions for drone k in the *Update-Subtour()* function on line 6. Observe that we can form a closed cycle out of each drone sortie $\mathcal{T}_k$ by connecting the *take-off* and *land* locations. We can treat $\mathcal{T}_k$ as a Traveling Salesman Problem (TSP), a well studied problem with both complete and approximate solutions [27], by setting the set $V = \{{}^s a_k, {}^e a_k\} \cup I_k$ as vertices, $E = V \times V$ as edges and fixing the edge $({}^e a_k, {}^s a_k)$ to be in the solution. The *Update-Subtour()* function improves each set of drone actions by transforming each sortie into this special instance of a TSP and solving it using a polynomial time TSP solver [9].

After running the *Update-Subtour()* function for each drone k assigned to UGV j, we optimize the location of each launch and landing action pairs in $\mathcal{S}'$ while adhering to the energy limitations and recharging time for each drone on the UGV. This is done in the *optimize-launch-land()* function on line 8 via a SOC program, which is further discussed in Section 4.3.

Algorithm 1 Iterative Launch Optimizer Algorithm

Input: PoI set I, drone set R^a, UGV set R^g, drone-to-UGV set D_j
Output: Patrolling kernel $\mathcal{S}$

1: $\mathcal{S} \leftarrow$ *Initial-Solution*(I, R^a, R^g, D_j)
2: **repeat**
3: $\quad \mathcal{S}' \leftarrow \mathcal{S}$
4: $\quad$ **for each** $j \in R^g$ **do**
5: $\quad\quad$ **for each** $k \in D_j$ **do**
6: $\quad\quad\quad \mathcal{S}' \leftarrow$ *Update-Subtour*$(\mathcal{S}', k)$
7: $\quad\quad$ **end for**
8: $\quad\quad \mathcal{S}' \leftarrow$ *optimize-launch-land*$(\mathcal{S}', j)$
9: $\quad$ **end for**
10: $\quad$ **if** $PAR(\mathcal{S}') < PAR(\mathcal{S})$ **then**
11: $\quad\quad \mathcal{S} \leftarrow \mathcal{S}'$
12: $\quad$ **end if**
13: **until** $\mathcal{S}'$ does not change

If the updated interim solution $\mathcal{S}'$ reduces PAR from the previous solution $\mathcal{S}$, then we accept the interim solution as the new solution on line 11 and repeat steps 2 through 13. This procedure repeats until we no longer see an improvement in the interim solution $\mathcal{S}'$.

As an alternative to the ILO algorithm, the LO algorithm finds an initial solution using the *Initial-Solution()* function then runs the *optimize-launch-land()* function exactly once for each UGV and ends. The LO algorithm does not run the *Update-Subtour()* function.

4.2 Finding an Initial Solution

We use a modified version of the algorithm in [25] to find an initial solution. The original algorithm runs as follows: (1) break up the environment into sub-regions using predetermined region sizes, (2) order sub-regions using a TSP solver, (3) order and divide discrete nodes by their quadrant angles from the centroid of the sub-region, and (4) order nodes for each drone using a TSP solver. We modify this approach by using a VRP solver for steps 1 and 2 then use a VRP solver in each sub-region to accomplish steps 3 and 4. We also incorporate UGV energy limitations into the algorithm.

Listing 2 provides details of the *Initial-Solution()* function. The algorithm forms $c = m_g$ clusters using Lloyd's k-means clustering algorithm on line 3. After finding PoI clusters C, the algorithm treats the centroids of the PoI clusters in C as stops for a Vehicle Routing Problem with $m' = m_g$ vehicles [28], which is solved using the *VRP()* function on line 4. To implement a VRP solver in the *VRP()* function, we further divided the PoI centroids into m' clusters and then solve a TSP on each cluster using the Lin-Kernighan-Helsgaun TSP solver [9], though this method could be swapped out with other VRP solvers such as [5] and the rest of the *Initial-Solution()* function algorithm would remain the same. Figure 3 shows an example of the PoI clustering and VRP for the UGV steps in panel 2.

For each cluster C' assigned to UGV j, the algorithm solves another VRP on line 7 that sends each drone assigned to j to visit the PoI in C' using the centroid of the cluster as the depot for the VRP (where the UGV will wait while each drone visits PoI in C'). After finding a set of drone sorties $\mathcal{T}^a$ for cluster C', the algorithm verifies that none of the drone sorties $\mathcal{T}_k \in \mathcal{T}^a$ violate drone energy constraints. If there is a drone sortie that exceeds the drone's energy limit, the algorithm increments c and restarts. Figure 3 demonstrates finding drone sorties on a PoI cluster while the UGV waits at the cluster centroid in panel 3. If there exists a UGV tour that runs a UGV out of energy, then the algorithm increments m' by m_g and restarts. Otherwise, the algorithm forms action sequences that form a patrolling kernel to the Team-PITH problem.

Algorithm 2 Initial-Solution Algorithm

Input: PoI set I, drone set R^a, UGV set R^g, drone-to-UGV set D_j
Output: Patrolling kernel $\mathcal{S}$

1: $m' \leftarrow |R^g|$, $c \leftarrow m'$
2: $c \leftarrow \max(c, m')$
3: $C \leftarrow k\text{-}means(c)$
4: $\mathcal{T}^g \leftarrow VRP(C, m')$
5: **for each** $j \in R^g$ **do**
6: **for each** cluster C' assigned to j in $\mathcal{T}^g$ **do**
7: $\mathcal{T}^a \leftarrow VRP(C', D_j)$
8: **if** $\exists \mathcal{T}_k \in \mathcal{T}^a$ such that dist$(\mathcal{T}_k) >$ ${}^{max}d_k$ **then**
9: $c \leftarrow c + 1$, repeat steps 2–8
10: **end if**
11: **end for**
12: **end for**
13: **if** $\exists \mathcal{T}_j \in \mathcal{T}^g$ such that energy$(\mathcal{T}_j) >$ ${}^{max}d_j$ **then**
14: $m' \leftarrow m' + |R^g|$, repeat steps 2–13
15: **end if**
16: $A^g, A^a \leftarrow Form\text{-}Actions(\mathcal{T}^g, \mathcal{T}^a)$
17: **return** $\mathcal{S} = \{A^g, A^a\}$

4.3 Optimizing Launch and Land Actions

The *optimize-launch-land()* function is given sets of actions for a drone-UGV team and optimizes the location of each of these actions via a SOC program while holding a set of assumptions. The function returns optimal action locations, within the limitations of the assumptions. These assumptions are:

(1) the order of drone launch and land actions is known a priori,
(2) the drones must recharge completely between consecutive sorties,
(3) vehicle position accuracy when visiting any given PoI or the base station can be relaxed to some constant ϵ, and
(4) the travel distance limits of any vehicle can be relaxed by ϵ.

The *Initial-Solution()* function assigns an ordering to each drone sortie, giving us assumption 1. Our *Initial-Solution()* function pushes each drone as close as it can to the limit of their operating distance, meaning that each drone is expected to need a near-full recharge, validating assumption 2. Furthermore, standard commercially available GPS have limited accuracy, validating assumption 3, and the distance that each vehicle will be traveling is far enough that assumption 4 should not impact the validity of a solution.

Given a patrolling kernel $\mathcal{S}$, we formulate a SOC program using the following variables: For each sortie $\mathcal{T}_k \in \mathcal{S}$, we create continuous variables (x_i, y_i) and (x_j, y_j) for the location of the launch and land actions, ${}^s a_k$ and ${}^e a_k \in \mathcal{T}_k$, respectively. Let $({}^s x_k, {}^s y_k)$ and $({}^e x_k, {}^e y_k)$ be the location of the first and last PoI in $I_k \in \mathcal{T}_k$, respectively, which are continuous variables that we constrain to stay within an $\epsilon \times \epsilon$ box centered on their respective PoI. Let ${}^t d_k$ be the distance required to traverse sequence $I_k \in \mathcal{T}_k$, which we assume to be constant. Let continuous variables ${}^s d_k$ and ${}^e d_k$ represent the distance from the launch action, ${}^s a_k$, and the land action, ${}^e a_k$, to the first and last PoIs in I_k, respectively. Let ${}^a t_i$ be the time required to perform action $a_i \in A^g_j$ and t_k be the time required for drone k to land, both of which are constant. We use t_i to track the end time for each action $a_i \in A^g_j$. Let variable t_b specifically be the time the "end" action is completed (the time that the UGV returns to the base station). Let d_j be the distance from a_i to a_j, for consecutive actions a_i and $a_j \in A^g_j$.

We formulate a SOC program to optimize the location of the launch and land actions as follows:

$$\min t_b \tag{11}$$

Subject to

$$t_i \geq t_j + {}^{max\text{-}charge}t_k, \forall \text{ launch } a_i \text{ after land } a_j \in \tilde{A}^a \tag{12a}$$

$$t_{i+1} \geq t_i + \frac{1}{v_g} d_{i+1} + {}^{a}t_{i+1}, i \in \{1, 2, \cdots k-1\} \quad (13a)$$

$$t_j = t_i + \frac{1}{v_k}({}^{s}d_k + {}^{t}d_k + {}^{e}d_k) + t_r, \forall\, {}^{s}a_k, {}^{e}a_k \in \mathcal{T}_k \quad (13b)$$

$${}^{s}d_k^2 \geq (x_i - {}^{s}x_k)^2 + (y_i - {}^{s}y_k)^2, \forall \mathcal{T}_k \quad (13c)$$

$${}^{e}d_k^2 \geq (x_j - {}^{e}x_k)^2 + (y_j - {}^{e}y_k)^2, \forall \mathcal{T}_k \quad (13d)$$

$$d_j^2 \geq (x_i - x_j)^2 + (y_i - y_j)^2, i \in \{1, 2, \cdots k-1\}, j = i+1 \quad (13e)$$

$${}^{max}d_k \geq {}^{s}d_k + {}^{t}d_k + {}^{e}d_k, i \in \{1, 2, \cdots k-1\}, \forall \mathcal{T}_k \quad (13f)$$

For each drone sortie, constraint (12a) does not allow a drone to be launched on a consecutive sortie until after it has been fully recharged (enforcing assumption 2). Constraint (13a) prevents action a_{i+1} from completing until after action a_i completes plus the time required for the UGV to move from a_i to a_{i+1} and perform action a_{i+1}. Constraint (13b) forces the end time of a drone sortie to be equal to the end time of the sortie's launch action plus the time required for the drone to traverse the distance of the sortie and land again. Constraints (13c) and (13d) control the distance of the start and end leg of each sortie, respectively, while constraint (13e) controls the total distance from one action to the next in A_j^g. Finally, constraint (13f) enforces each drone's energy constraint.

Observe that constraints (13c), (13d), and (13e) are convex quadratic constraints while all other constraints and the objective are linear. Therefore, Objective (11) with constants (12a) through (13f) form a SOC program, which can be solved in polynomial time to optimality using interior point and the barrier method [4].

4.4 Time Complexity Analysis

The *Initial-Solution()* function will run the *VRP()* function n^2 times in the worst case. The *VRP()* function depends on the LKH solver, which has a time complexity of $O(n^{2.2})$ [8]. If we limit the number of times that Lloyd's algorithm iterates to some constant, then the *Initial-Solution()* function has a runtime of $O(n^{4.2})$.

The efficiency of the *optimize-launch-land()* function depends on the performance of the solver used, though we know for certainty that the SOC program can be solved in polynomial time [20]. Let ζ be the time required to run the *optimize-launch-land()* function. The LO algorithm will run the *Initial-Solution()* function once and the *optimize-launch-land()* function m_g times, giving it a time complexity of $O(n^{4.2} + m_g\zeta)$. If we set an iteration limit of φ on the loop starting at line 2 in the ILO algorithm, then the algorithm will run the *Update-Subtour()* function φm_a times and the *optimize-launch-land()* function φ times. The *Update-Subtour()* function again depends on the LKH solver, which has a runtime of $O(n^{2.2})$. This gives the ILO algorithm a time complexity of $O(n^{4.2}+\varphi(m_g\zeta+n^{2.2}))$. For both algorithms, we expect the number of PoI to be much larger than the number of UGV ($n \gg m_g$) so we should expect the $n^{4.2}$ term from the *Initial-Solution()* function to dominate the runtime.

5 NUMERICAL SIMULATION RESULTS

We created three test series to evaluate our proposed algorithms. In the first series, termed the *IncreasingNodes* tests, we have a fixed team (1 UGV, 2 drones) and increase the number of PoIs, from five up to 100 in increments of five, with 50 uniquely generated tests at each increment. In the second series, termed the *IncreasingDrone* tests, we have 50 random PoIs and a single UGV with an increasing number of drones, from one up to 10 drones in increments of one with 50 unique tests at each increment. Our final test series has 50 random PoIs and increases the number of drone-UGV teams, from one team up to 10 in increments of one, where each team consists of a 1:2 ratio of UGVs to drones. We generated 50 tests at each increment and term this the *IncreasingTeams* tests.

As a baseline method, we use the *Initial-Solution()* function by itself. This function is a modification of the proposed algorithm in [13, 25], which was used for a closely related patrolling problem, and is similar to the proposed solutions in [14, 22].

All algorithms were implemented in C++. We used Gurobi version 11.0.3 to solve the SOC program and LKH version 3.0.8. To promote good research integrity, the source code to our algorithm has been made open source[1].

5.1 Algorithm Performance

We ran each algorithm and the baseline on the generated test series to evaluate their performance. We recorded both the PAR and the Worst Latency (in hours) metrics.

Figure 4 shows the results for each algorithm on the *IncreasingNodes* tests series. The ILO algorithm shows an average PAR

[1] www.github.com/pervasive-computing-systems-group/HitchhikerPatrollingSolver

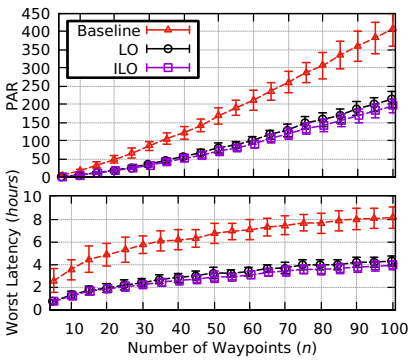

Figure 4: *IncreasingNodes* test series.

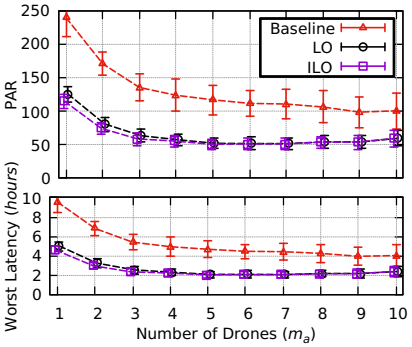

Figure 5: *IncreaseDrones* test series.

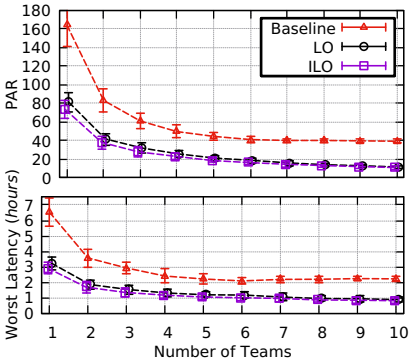

Figure 6: *IncreaseTeams* test series.

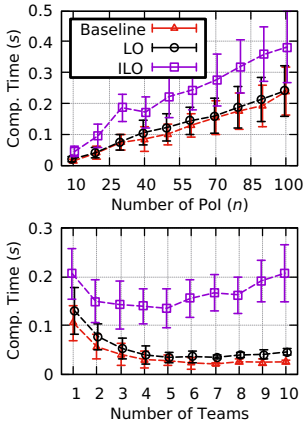

Figure 7: Computation times.

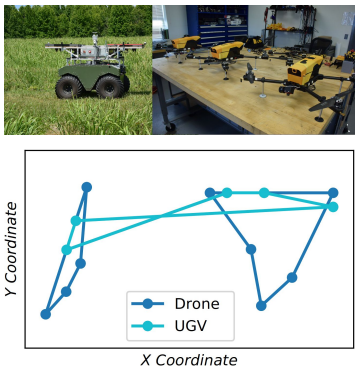

Figure 8: Prototype setup.

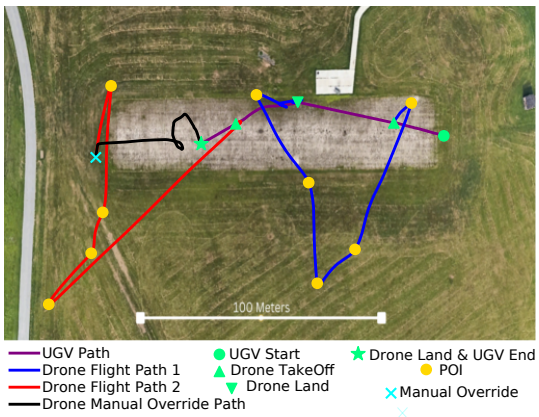

Figure 9: Drone and UGV paths from field experiment.

reduction of 57.7% and 7.5% against the baseline and LO algorithms, respectively. The LO algorithm has a PAR reduction of 54.1% compared to the baseline. Results from the *IncreasingDrone* test series show similar trends (shown in Figure 5). The ILO algorithm has an average PAR reduction of 52.0% and 3.4% compared to the baseline and LO algorithms, respectively. The LO algorithm recorded a 50.3% PAR reduction compared to the baseline. Finally, Figure 6 shows the results of the *IncreasingTeams* test series. The ILO algorithm shows a PAR reduction of 60.6% and 9.1% compared to the baseline and LO algorithms, respectively. The LO algorithm records a PAR reduction of 56.4% compared to the baseline. We see the same performance trends in Worst Latency that we found in PAR.

5.2 Computation Time

Figure 7 shows computational results as the number of PoI increases (top graph) and as the number of teams increases (bottom graph). As the number of PoI increases, the ILO algorithm has an increased average computation time factor of 3.8 and 2.1 relative to the baseline and LO algorithms, respectively, while the LO algorithm has an average computation time factor of 1.9 higher than the baseline.

As the number of teams increases, the ILO algorithm runs with an average factor of 9.5 and 5.6 slower than the baseline and LO, respectively, and the LO algorithm runs with an average slowdown factor of 1.7 relative to the baseline. We note that the computation time for the LO and baseline approach dropped as the number of teams increased because having more teams to deploy makes it easier to find an initial solution, which reduces the number of times Algorithm 2 must solve an instance of the VRP.

5.3 Results Summary

Our results show that the LO and ILO algorithms produce statistically significant reductions in both PAR and Worst Latency when compared against the baseline. We also found a tradeoff in performance and computation time between the LO and ILO algorithms. The ILO algorithm achieves lower PAR compared to the LO algorithm but is more computationally demanding.

The number of PoI has the largest impact on computation time. The LO and ILO algorithms both have a time complexity of $O(n^{4.2})$ and therefore scale similarly with increasing input size. However, repeatedly running the TSP solver in the ILO algorithm has a noticeable impact on computation time.

6 PROTOTYPE ON REAL DRONE-UGV TEAM

To demonstrate the Team-PITH problem and to determine limitations of our proposed algorithm, we ran a field prototype of the considered problem scenario on a physical UGV and drone testbed. We used a Clearpath™ Warthog and a custom built drone platform. The Warthog was configured to wirelessly transfer energy to the drone. We set a series of PoIs in an empty field and had the Warthog ferry the drone to different launch points.

Figure 8 shows the robots and plan generated by the ILO algorithm and Figure 9 shows the GPS trace of each vehicle. The drone traveled all assigned routes successfully but the Warthog became stuck halfway through its assigned route due to issues with its local planner and did not reach the final rendezvous location. As a result, the drone had to be manually flown back to the Warthog after the second sortie. However, the first drone sortie provides a clear example of the Team-PITH problem. Possible solutions to deployment issues include strengthening information sharing between the UGV and drone and online replanning.

7 CONCLUSIONS

In this article, we introduced the Team-PITH problem and presented the Launch Optimizer (LO) and Iterative Launch Optimizer (ILO) algorithms. Given an initial solution, the LO algorithm uses a Second-Order Cone program to find locally optimal solutions, which can be solved in polynomial time, while the ILO algorithm iteratively runs the steps of the LO algorithm mixed with a Traveling Salesman Problem. We demonstrate through experimental data that the LO and ILO algorithms reduce PAR values by over 50% as compared to a baseline method from existing literature.

Limitations of this work include ignoring ground obstacles, assuming fixed drone-UGV teams, and assumptions on the reliability of robot movement (as identified in our field experiments). For future work, we plan to address these limitations (particularly avoiding ground obstacles and online adaptive planning) as well as handling limited communication and varying PoI priorities.

REFERENCES

[1] United States Environmental Protection Agency. 2007. *Leak Detection and Repair: A Best Practices Guide.* Office of Compliance, Office of Enforcement and Compliance Assurance, Washington, DC. https://www.epa.gov/compliance/leak-detection-and-repair-best-practices-guide Accessed: 2024-11-27.

[2] Ahmad Bilal Asghar, Guangyao Shi, Nare Karapetyan, James Humann, Jean-Paul Reddinger, James Dotterweich, and Pratap Tokekar. 2023. Risk-aware recharging rendezvous for a collaborative team of uavs and ugvs. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 5544–5550.

[3] Nicola Basilico. 2022. Recent trends in robotic patrolling. *Current Robotics Reports* 3, 2 (2022), 65–76.

[4] Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization* (1st ed.). Cambridge University Press.

[5] Ilya Builuk. 2023. A new solver for rich Vehicle Routing Problem. https://doi.org/10.5281/zenodo.4624037

[6] Jonathan Diller, Peter Hall, and Qi Han. 2023. Holistic path planning for multi-drone data collection. In *International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*. IEEE, 222–226.

[7] Jonathan Diller and Qi Han. 2023. Energy-aware UAV Path Planning with Adaptive Speed. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*. 923–931.

[8] Keld Helsgaun. 2000. An effective implementation of the Lin–Kernighan traveling salesman heuristic. *European journal of operational research* 126, 1 (2000), 106–130.

[9] Keld Helsgaun. 2017. An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University* 12 (2017), 966–980.

[10] Arnon M Hurwitz, James M Dotterweich, and Trevor A Rocks. 2021. Mobile robot battery life estimation: battery energy use of an unmanned ground vehicle. In *Energy Harvesting and Storage: Materials, Devices, and Applications XI*, Vol. 11722. SPIE, 24–40.

[11] Nare Karapetyan, Ahmad Bilal Asghar, Amisha Bhaskar, Guangyao Shi, Dinesh Manocha, and Pratap Tokekar. 2024. Ag-cvg: Coverage planning with a mobile recharging ugv and an energy-constrained uav. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2617–2623.

[12] Richard M Karp. 2010. *Reducibility among combinatorial problems.* Springer.

[13] Xiaoshan Lin, Yasin Yazıcıoğlu, and Derya Aksaray. 2022. Robust planning for persistent surveillance with energy-constrained UAVs and mobile charging stations. *IEEE Robotics and Automation Letters* 7, 2 (2022), 4157–4164.

[14] Yao Liu, Zhihao Luo, Zhong Liu, Jianmai Shi, and Guangquan Cheng. 2019. Cooperative routing problem for ground vehicle and unmanned aerial vehicle: The application on intelligence, surveillance, and reconnaissance missions. *IEEE Access* 7 (2019), 63504–63518.

[15] Zhilong Liu, Raja Sengupta, and Alex Kurzhanskiy. 2017. A power consumption model for multi-rotor small unmanned aircraft systems. In *international conference on unmanned aircraft systems (ICUAS)*. IEEE, 310–315.

[16] Parikshit Maini, Pratap Tokekar, and PB Sujit. 2020. Visibility-based persistent monitoring of piecewise linear features on a terrain using multiple aerial and ground robots. *Transactions on Automation Science and Engineering* 18, 4 (2020), 1692–1704.

[17] Parikshit Maini, Kevin Yu, PB Sujit, and Pratap Tokekar. 2018. Persistent monitoring with refueling on a terrain using a team of aerial and ground robots. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 8493–8498.

[18] Md Safwan Mondal, Subramanian Ramasamy, James D Humann, James M Dotterweich, Jean-Paul F Reddinger, Marshal A Childers, and Pranav Bhounsule. 2024. A Robust UAV-UGV Collaborative Framework for Persistent Surveillance in Disaster Management Applications. In *International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 1239–1246.

[19] Md Safwan Mondal, Subramanian Ramasamy, James D Humann, Jean-Paul F Reddinger, James M Dotterweich, Marshal A Childers, and Pranav A Bhounsule. 2023. Cooperative Multi-Agent Planning Framework for Fuel Constrained UAV-UGV Routing Problem. *arXiv preprint arXiv:2309.03397* (2023).

[20] Yurii Nesterov and Arkadii Nemirovskii. 1994. *Interior-Point Polynomial Algorithms in Convex Programming.* Society for Industrial and Applied Mathematics.

[21] Subramanian Ramasamy, Md Safwan Mondal, James D Humann, James M Dotterweich, Jean-Paul F Reddinger, Marshal A Childers, and Pranav A Bhounsule. 2024. Optimizing Routes of Heterogenous Unmanned Systems using Supervised Learning in a Multi-Agent Framework: A computational study. In *International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 286–294.

[22] Subramanian Ramasamy, Jean-Paul F Reddinger, James M Dotterweich, Marshal A Childers, and Pranav A Bhounsule. 2021. Cooperative route planning of multiple fuel-constrained Unmanned Aerial Vehicles with recharging on an Unmanned Ground Vehicle. In *International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 155–164.

[23] Jürgen Scherer and Bernhard Rinner. 2017. Short and full horizon motion planning for persistent multi-UAV surveillance with energy and communication constraints. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 230–235.

[24] Jürgen Scherer and Bernhard Rinner. 2020. Multi-UAV surveillance with minimum information idleness and latency constraints. *Robotics and Automation Letters* 5, 3 (2020), 4812–4819.

[25] Sepehr Seyedi, Yasin Yazicioğlu, and Derya Aksaray. 2019. Persistent surveillance with energy-constrained uavs and mobile charging stations. *IFAC-PapersOnLine* 52, 20 (2019), 193–198.

[26] Feng Shan, Junzhou Luo, Runqun Xiong, Wenjia Wu, and Jiashuo Li. 2020. Looking before crossing: An optimal algorithm to minimize UAV energy by speed scheduling with a practical flight energy model. In *INFOCOM - Conference on Computer Communications*. IEEE, 1758–1767.

[27] Steven S. Skiena. 2008. *The Algorithm Design Manual* (2nd ed.). Springer Publishing Company, Incorporated.

[28] Paolo Toth and Daniele Vigo. 2002. *The vehicle routing problem.* SIAM.

[29] Wei Wei, Shuiyuan Cheng, Guohao Li, Gang Wang, and Haiyan Wang. 2014. Characteristics of volatile organic compounds (VOCs) emitted from a petroleum refinery in Beijing, China. *Atmospheric Environment* 89 (2014), 358–366.

[30] Chuanbo Yan and Tao Zhang. 2016. Multi-robot patrol: A distributed algorithm based on expected idleness. *International Journal of Advanced Robotic Systems* 13, 6 (2016), 1729881416663666.

[31] Kevin Yu, Ashish Kumar Budhiraja, Spencer Buebel, and Pratap Tokekar. 2019. Algorithms and experiments on routing of unmanned aerial vehicles with mobile recharging stations. *Journal of Field Robotics* 36, 3 (2019), 602–616.

[32] Kevin Yu, Jason M O'Kane, and Pratap Tokekar. 2019. Coverage of an environment using energy-constrained unmanned aerial vehicles. In *international conference on robotics and automation (ICRA)*. IEEE, 3259–3265.

[33] Yong Zeng, Jie Xu, and Rui Zhang. 2019. Energy minimization for wireless communication with rotary-wing UAV. *transactions on wireless communications* 18, 4 (2019), 2329–2345.

[34] Juan Zhang, James F Campbell, Donald C Sweeney II, and Andrea C Hupman. 2021. Energy consumption models for delivery drones: A comparison and assessment. *Transportation Research Part D: Transport and Environment* 90 (2021), 102668.