Improved Algorithms for Co-Scheduling of Edge Analytics and Routes for UAV Fleet Missions

Aakash Khochare, Francesco Betti Sorbelli, *Member, IEEE*, Yogesh Simmhan, *Senior Member, IEEE*, and Sajal K. Das, *Fellow, IEEE*

***Abstract*— Unmanned Aerial Vehicles (UAVs) or drones are increasingly used for urban applications like traffic monitoring and construction surveys. Autonomous navigation allows drones to visit *waypoints* and accomplish *activities* as part of their *mission*. A common activity is to hover and observe a location using on-board cameras. Advances in Deep Neural Networks (DNNs) allow such videos to be analyzed for automated decision making. UAVs also host edge computing capability for on-board inferencing by such DNNs. To this end, for a fleet of drones, we propose a novel *Mission Scheduling Problem (MSP)* that co-schedules the flight routes to visit and record video at waypoints, and their subsequent on-board edge analytics. The proposed schedule maximizes the data capture and computing utilities from the activities while meeting the activity deadlines, and the energy and computing constraints. We first prove that MSP is NP-hard and then optimally solve it by formulating a mixed integer linear programming (MILP) problem. Next, we design five time-efficient heuristic algorithms that provide sub-optimal but fast solutions that are empirically competitive with the optimal solution. Evaluation of these five schedulers using real drone traces demonstrate utility–runtime trade-offs under diverse workloads.**

***Index Terms*— UAV, drone, edge computing, vehicle routing, job scheduling, energy constrained, video analytics.**

I. Introduction

UNMANNED Aerial Vehicles (UAVs), also called *drones*, are enabling a wide range of applications in smart cities [1], such as traffic monitoring [2], construction surveys [3], package delivery [4], localization [5], and disaster management [6], assisted by 5G wireless roll-out [7]. The mobility, agility, and hovering capabilities of drones allow them to rapidly fly to points of interest (i.e., *waypoints*) in the city to accomplish specific *activities*, e.g., observing traffic at hot-spots during commute hours, status of building construction, or crowding among pedestrians during COVID-19. Usually, such activities involve hovering and recording a scene using the drone's camera, and analyzing the videos to take decisions, such as changing traffic signaling, flagging construction delays, encouraging pedestrians to practice social distancing, etc.

Advancements of computer vision algorithms and *Deep Neural Networks* (DNNs) enable video analytics to be performed over such recordings for automated decision-making, which are inferred once these are transferred to a ground station (GS) after the drones land. In-flight transfer of videos to a GS is limited by the intermittent bandwidth of current communications technologies. However, certain activities may require low-latency analysis and decisions, as soon as the video is captured at a location. So, the on-board *edge computing* capability (like ARM CPUs and NVIDIA Jetson GPUs [8]) on commercial drones can be leveraged to process the recorded videos, and quickly report concise results to the GS over 4/5G networks [9]. Since the transferred results are brief and the *on-board* processing times dominate, we ignore communication issues like data rate, latency, and reliability that are affected by the UAV's altitude, antenna envelope, etc.

UAVs are *energy-constrained vehicles* with limited battery capacity, and commercial drones can currently fly for less than an hour. The flying distance between waypoints will affect the number of activities that can be completed in one *trip* on a full battery. Besides hovering and recording videos at waypoints, performing edge analytics also consumes energy. So, the drone's battery capacity should be judiciously managed for the flying, hovering, and computing tasks. Nevertheless, once a drone lands, its battery can be quickly replaced with a full one, to be ready for a new trip.

This paper examines how a *UAV fleet operator* in a city can plan *missions* for a captive set of drones to accomplish activities periodically provided by the users. An *activity* involves visiting a waypoint, hovering and capturing video at that location for a specific time period, and optionally performing on-board analytics on the captured data. Activities also offer *utility* scores depending on how they are handled. The novel problem we propose here is for the fleet operator to *co-schedule flight routing among waypoints and on-board computation so that the drones complete (a subset of) the provided activities, within the energy and computation constraints of each drone, while maximizing the total utility.*

Existing works have examined routing of one or more drones for capturing and relaying data to the backend [10], off-loading computations from mobile devices [11], cooperative video surveillance [12]. There also exists literature on scheduling tasks for edge computing that are compute- and energy-aware, operate on distributed edge resources, and consider deadlines and device reliability [13]. None of these examine co-scheduling a fleet of physical drones and digital applications on them to meet the objective, while efficiently managing the energy capacity to maximize utility.

Manuscript received 9 March 2022; revised 15 February 2023; accepted 14 April 2023; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor A. Khreishah. Date of publication 8 June 2023; date of current version 16 February 2024. This work was supported in part by NSF under Award CSSI-2104078, Award SCC-1952045, and Award OAC-1725755; and in part by "Gruppo Nazionale per il Calcolo Scientifico (GNCS)—Istituto Nazionale di Alta Matematica (INdAM)." *(Corresponding author:Yogesh Simmhan.)*

Aakash Khochare and Yogesh Simmhan are with the Department of Computational and Data Sciences, Indian Institute of Science, Bengaluru 560012, India (e-mail: simmhan@iisc.ac.in).

Francesco Betti Sorbelli is with the Department of Computer Science and Mathematics, University of Perugia, 06121 Perugia, Italy.

Sajal K. Das is with the Department of Computer Science, Missouri University of Science and Technology, Rolla, MO 65409 USA.

Digital Object Identifier 10.1109/TNET.2023.3277810

Specifically, our *Mission Scheduling Problem (MSP)* combines elements of the *Vehicle Routing Problem (VRP)* [14], which generalizes the well known Traveling Salesman Problem (TSP) to find optimal routes for a set of vehicles and customers [15], and the *Job-shop Scheduling Problem (JSP)* [16] for mapping jobs of different execution duration to the available resources, which is often used for parallel scheduling of computing tasks to multiprocessors [17].

This paper substantially extends our prior conference paper [18]. There, we formulated the MSP and devised the MILP-based OPT solution, besides two heuristics: RSS, which gave good utility but had a high computational cost, and CS, which was time-efficient but with a lower utility. Here, we add *three new algorithms* for solving the MSP previously introduced — EOFO which we find generally performs better than the other heuristics on both utility and time, HUFD which works well when the utility of activities are unbalanced, and IPS that performs well when the number of activities is much larger than the number of drones. In particular, EOFO and IPS provide much better results than the previously published CS [18] due to their activity selection rules being optimized better for the task-scheduling sub-problem. Further, our experimental comparison is much more rigorous, with *seven additional diverse workloads* evaluated for all six algorithms, and an analysis of which ones perform well under what conditions.

We make the following contributions in this article.

- We characterize the system and application model, and formally define the *Mission Scheduling Problem (MSP)* to co-schedule routes and analytics for a fleet of drones, maximizing the obtained utility.
- After proving that MSP is an *NP-hard* problem, we optimally solve it using a *mixed integer linear programming (MILP)* design, OPT, which is feasible for small inputs due to its intrinsic complexity.
- For solving the MSP for arbitrary-sized inputs, we design *five time-efficient heuristic algorithms*, i.e., CS that is in average the best performing algorithm, RSS that is the fastest in practice, HUFD that works well when the utility of activities are pretty unbalanced, and EOFO and IPS that are suitable when the number of activities is large with respect to the number of drones. We also offer time complexity bounds for them.
- We *evaluate and analyze* the utility and scheduling runtime trade-offs for these algorithms, for diverse drone workloads based on real drone traces, i.e., *Random* (RND) and *Depth First Search* (DFS). The evaluation shows that our presented heuristics are able to obtain in average at least the 60-65% of the total utility in reasonable time.

The rest of the paper is organized as follows. In Section II we present the related work; Section III provides the system and application model, and assumptions; Section IV formalizes our MSP; Sections V and VI offer optimal and heuristic algorithms to solve MSP; Section VII reports experimental results; finally, Section VIII concludes the paper.

II. Related Work

This section reviews literature on vehicle routing and job-shop scheduling, contrasting them with our MSP.

A. Problems Concerning Vehicles to Route

The Traveling Salesman Problem (TSP) is a classic combinatorial problem that aims at finding the shortest route along edges that visits all the vertices in a given graph. Specifically, given a weighted, undirected and complete graph with a set of vertices and a set of edges, where each edge has a weight that characterizes the travel distance, TSP asks to find the minimum weight (and so the "shortest") route in the graph such that every vertex is visited exactly once. This problem has been proven to be NP-hard [19], and many different heuristics have been proposed in the literature. There is an efficient 2-approximate solution that assume triangle inequality among edge weights [20]. The more sophisticated Christofides Algorithm lowers this approximation ratio to $\frac{3}{2}$ [21]. A complete, recent survey on TSP and its variants can be found in [22].

VRP is a variant of TSP with multiple salespersons [14] and it is NP-hard [23]. This problem has had several extensions to handle realistic scenarios, such as temporal constraints that impose deliveries only at specific time-windows [24], capacity constraints on vehicle payloads [25], multiple trips for vehicles [26], profit per vehicle [27] and traffic congestion [28]. VRP has also been adapted for route planning for a fleet of ships [29], and for drone delivery [30].

In [10] the scheduling of *events* is performed by UAVs at specific locations, involving data sensing/processing and communication with the GS. The goal is to minimize the drone's energy consumption and operation time. Factors like wind that may affect the route and execution time are also considered. While they combine sensing and processing into one monolithic event, these are actually independent tasks which need to be co-scheduled, as we do. Also, they minimize the operating time and energy while we maximize the utility to perform tasks within a time and energy budget.

Others [11] explore the use of UAVs to off-load computing from the users' mobile devices, and for relaying data between mobile devices and GS. The authors considered the drones' trajectory, bandwidth, and computing optimizations in an iterative manner. The aim is to minimize energy consumption of the drones and mobile devices. It is validated through simulation for four mobile devices. We instead consider a more practical problem for a fleet of drones with possibly hundreds of locations to visit and on-board computing tasks to perform, and validate through simulation traces for up to 100 drones.

B. Problems Concerning Tasks to Schedule

Job Shop Scheduling (JSP) is classic optimization problem. Given n jobs that have varying processing time, the problem involves finding a scheduling across m machines such that the makespan is minimized. The problem is known to be NP-Hard. Several extensions have been proposed for the JSP problem such as deadlines [31], failures [32], and preemption [33]. Scheduling of computing tasks on drones is closely aligned with scheduling tasks on edge and fog devices [34] and mobile edge computing (MEC) [35], and broadly with parallel workload scheduling [17] and JSP [16].

In [13], an online algorithm is proposed for deadline-aware task scheduling for edge computing. It highlights that workload scheduling on the edge has several dimensions, and it jointly optimizes networking and computing to yield the best possible schedule. Feng at al. [36] propose a framework for cooperative edge computing on autonomous road vehicles, which aims to increase their decentralized computational capabilities and task execution performance.

There exist works that explore task scheduling for mobile users and off-load computing to nearby edge/fog resources.

These may be categorized based on their use of *predictable* or *unpredictable* mobility models. In [37], the mobility of a vehicle is predicted and used to select the road-side edge computing unit to which the computation is off-loaded. Serendipity [38] takes an alternate view and assumes that mobile edge devices interacts with each other intermittently and at random. This makes challenging to determine if tasks should be off-loaded to another proximate device for reliable completion. Our problem is complementary and does not involve off-loading. The possible waypoints are known ahead, and we perform predictable UAV route planning and scheduling of the computing locally on the edge.

Scheduling on the energy-constrained edge has also been investigated by Zhang et al. [39], where an energy-aware off-loading scheme is proposed to jointly optimize communication and computation resource allocation on the edge, and to limit latency. Our proposed problem also considers energy for the drone flight while meet deadlines for on-board computing. Others [40] propose a framework for cooperative edge computing on autonomous road vehicles, aimed at increasing their computational capabilities in a decentralized manner.

Several works [41], [42] have explored the use of *Deep Reinforcement Learning (DRL)* for UAV path planning. Ferdowsi et al. [43] formulate an MILP to optimize for the drone trajectory and the freshness of information. They propose a Deep Q-network and and LSTM based autoencoder to solve the formulation. Deliversense [44] looks at joint optimization of crowdsensing and deliveries by drones, here again formulated as an MILP and solved using DRL. While RL techniques benefit from good accuracy and generalization to unseen environments [45], they require prior training data, unlike heuristics. In future, heuristics and DRL techniques can complementing each other to solve such scheduling problems.

Similarly, *compressive sensing* has been used to reduce the dimensionality of sparse data, and obtain a reduced representation using fewer measurements than the original data [46]. This dimensionality reduction can significantly reduce the computational requirements to solve NP-hard optimization problems like ours. The key is to identify the sparsity patterns in the data fields obtained by the UAVs and then applying compressive sensing techniques while still preserving its essential information. Such alternatives can be explored in future.

III. Models and Assumptions

This section introduces the UAV system and application models along with our underlying assumptions.

A. UAV System Model

Let $\widehat{\lambda}$ be the *location* of a *UAV depot* in the city (see Figure 1, left) centered at the origin of a 3D Cartesian coordinate system. Let $D = \{d_1, \ldots, d_m\}$ be the set of m available homogeneous drones. Each drone has a camera for recording videos, which is subsequently processed. This processing can be done using the on-board computing, or offline once the drone lands (outside the scope of our problem). The on-board *processing speed* is π floating point operations per second (FLOPS). For simplicity, this is taken as cumulative across CPUs and GPUs on the drone, and this capacity is orthogonal to any computation done for navigation.

The battery on a drone has a fixed *energy capacity* E, which is used both for flying and for on-board computation. The drone's energy consumption has three components – *flying*, *hovering* and *computing*. Let ϵ^f be the energy required for flying for a unit time duration at a constant energy-efficient speed s within the city; let ϵ^h be the energy for hovering for a unit time duration; and let ϵ^c be the energy for performing computation for a unit time duration at full CPU/GPU utilization. For simplicity, we ignore the energy used for video capture since it is usually negligible [47], [48]. Also, a drone that returns to the depot can swap-in a fully charged battery pack and immediately start a new trip.

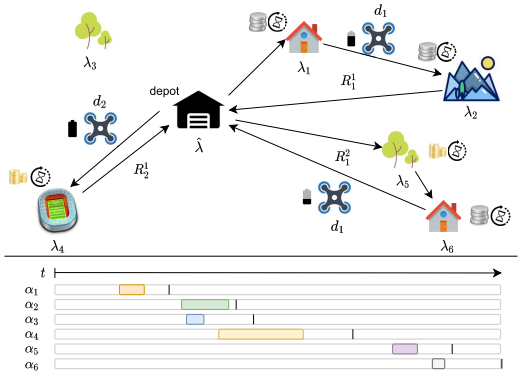


Fig. 1. Sample MSP scenario showing a city with the depot ($\widehat{\lambda}$); 6 waypoints to visit (λ_i) with some utility; and possible trip routes for drones (R_j^i); then, the corresponding 6 activities (α_i) with data capture duration (shaded) and compute deadline (vertical line) and 2 drones.

B. Application Model

Let $A = (\alpha_1, \ldots, \alpha_n)$ be the set of n activities to be performed starting from time $\widehat{t} = 0$, where each *activity* α_i is given by the tuple $\langle \lambda_i, t_i, \bar{t}_i, \kappa_i, \delta_i, \gamma_i, \bar{\gamma}_i, \bar{\bar{\gamma}}_i \rangle$. Here, $\lambda_i = (x_i, y_i, z_i)$ is the waypoint *location* coordinates where the video data for that activity has to be captured by the drone, relative to the depot location $\widehat{\lambda}$. The *starting and ending times* for performing the *data capture task* are t_i and $\bar{t}_i$. The *compute requirements* for processing all of the captured data is κ_i floating point operations. Lastly, δ_i is the *time deadline* by which the *computation task* should be completed on the drone to derive on-time utility of processing, while $\gamma_i, \bar{\gamma}_i$, and $\bar{\bar{\gamma}}_i$ are respectively the *data capture*, *on-time processing* and *on-board processing utility* values that are gained for completing the activity. These are described in the following subsection.

The computation may be performed incrementally on subsets of the video data, as soon as they are captured. This is common for analytics over constrained resources [49]. Specifically, for an activity α_i, the data captured between $(\bar{t}_i - t_i)$ is divided into *batches* of a fixed duration β, with the sequence of batches given by $B_i = (b_i^1, \ldots, b_i^{q_i})$, where $q_i = |B_i| = \left\lceil \frac{\bar{t}_i - t_i}{\beta} \right\rceil$. The computational cost to process each batch is $\kappa_i^k = \frac{\kappa_i}{q_i}$ FLOPS, and is constant for all batches of an activity. So, the *processing time* for the batch, given the processing speed π for a drone, is $\rho_i^k = \left\lceil \frac{\kappa_i^k}{\pi} \right\rceil$;[1] for simplicity, we discretize all time-units into integers.

C. Utility Model

The primary goal of drones is to capture videos at various locations for the specified duration. This is a *necessary* condition for an activity to be successful. We define this as the *data capture utility* (γ_i) accrued by a drone for an activity α_i.

[1] We assume full utilization of on-board CPU/GPU for processing.

The secondary goal is to opportunistically process the captured data using the on-board computing on the drone. Here, we have two scenarios. Some activities may not be time sensitive, and performing on-board computing is just to reduce the costs for offline computing. Here, processing the data captured by an activity using the drone's computing resources will provide an *on-board processing utility* ($\bar{\gamma}_i$). Other activities may be time-sensitive and have a *soft-deadline* δ_i for completing the processing. For these, if we process its captured data on the drone within this deadline, we receive an extra *on-time processing utility* ($\bar{\bar{\gamma}}_i$). The processing utilities accrue *pro rata*, for each batch of the activity completed.

D. Assumptions and Limitations

The application model is designed for periodic data collection and processing in applications like construction surveys and traffic monitoring, where the activities are known *a priori*. But, it does not support interactive applications or incremental activities, nor does it require timely transfer of collected data or processed results from the drone to the depot or the cloud. So, we do not impose any communications constraints.

Four properties of the application model, the activity *start and end time*, and the three utilities – *data capture*, *on-board processing* and *on-time processing* – can be used to model the urgency and priority of the data capture and processing. They indirectly capture the "Age of Information" metric as well [50]. Fairness across the activities is a non-goal.

Each scheduled activity is fully performed by one drone. The drones do not collaborate together for a single activity, where one drone could capture data and offload it to another drone (or the cloud) to process. Only one batch may be executed at a time on-board a drone and it is run to completion before scheduling another. There is no concurrent processing across batches or check-pointing of their progress. We do not allow data capture by a drone for multiple activities simultaneously. The data capture for an activity's batch may overlap with the computation of a previous batch of the same or a different activity. All batches for a single activity are executed in sequence, i.e., complete processing b_i^k before processing b_i^{k+1}. Once a batch is processed, its compact results are sent to the GS and accrue incremental utility value.

IV. Problem Formulation

The Mission Scheduling Problem (MSP) is summarized as: *Given a UAV depot in a city with a fleet of captive drones, and a set of observation and computing activities to be performed at locations in the city, each within a given time window and with associated utilities, the goal is to co-schedule the drones onto mission routes and the computation onto the drones, within the energy and compute constraints of the drones, such that the total utility achieved is maximized.* This is further formalized below.

A. Mission Scheduling Problem (MSP)

A UAV fleet operator receives and queues activities. Periodically, a mission schedule is planned to serve some or all the activities submitted so far using the whole fleet to maximize the operational utility that is gained. There may be other fixed costs for operating the captive fleet that we ignore.

Multiple activities can be assigned to the same drone d_j as part of the drone's *mission*, and the same drone can perform multiple *trips* from the depot for a mission. The *mission activities* for the r^{th} trip of a drone d_j is the ordered sequence $A_j^r = (\alpha_{j_1}^r, \ldots, \alpha_{j_n}^r) \subseteq A$ where $\alpha_{j_x}^r \in A$, $j_n \leq n$, and no activity appears twice within a mission. Further, we have $\alpha_{j_x}^r \prec \alpha_{j_{x+1}}^r \implies \bar{t}_{j_x}^r \leq t_{j_{x+1}}^r$, i.e., the observation start and end times of an activity in the mission sequence fully precede those of the next activity in it. Also, $A_j^x \cap A_k^y = \varnothing \; \forall j,k,x,y$ to ensure that an activity is assigned to just one drone. Depending on the feasibility and utility, some activities may not be part of any mission and are dropped, i.e., $\sum_j \sum_r |A_j^r| \leq n$. The *route* for the r^{th} trip of drone d_j is given by the ordered sequence $R_j^r = (\widehat{\lambda}, \lambda_{j_1}^r, \ldots, \lambda_{j_n}^r, \widehat{\lambda})$, where the starting and ending waypoints of the drone are the depot location $\widehat{\lambda}$, and each intermediate location corresponds to the video capture location $\lambda_{j_k}^r$ for the activity $\alpha_{j_k}^r$ in the mission sequence. For uniformity of notation, we denote the first and the last depot location in the route as $\lambda_{j_0}^r$ and $\lambda_{j_{n+1}}^r$, respectively.

A drone d_j, given the r^{th} trip of its route R_j^r, starts at the depot, visits each waypoint in the sequence and returns to the depot, where it may instantly get a fresh battery and start the $(r+1)^{th}$ route. Let drone d_j leave a waypoint location in its route, $\lambda_{j_i}^r$, at *departure time* $\tau_{j_i}^r$ and reach the next waypoint location, $\lambda_{j_{i+1}}^r$, at *arrival time* $\bar{\tau}_{j_{i+1}}^r$. Let the function $\mathcal{F}(\lambda_p, \lambda_q)$ give the *flying time* between λ_i and λ_j at a constant flying speed. So, we have $\bar{\tau}_{j_{i+1}}^r = \tau_{j_i}^r + \mathcal{F}(\lambda_{j_i}^r, \lambda_{j_{i+1}}^r)$. The drone must hover at each waypoint $\lambda_{j_i}^r$ between t_j^r and $\bar{t}_j^r$ while recording the video, and it departs the waypoint after this, i.e., $\tau_{j_i}^r = \bar{t}_{j_i}^r$. Also, if the drone arrives at this waypoint at time $\bar{\tau}_{j_i}^r$, i.e., before the observation start time t_j, it *hovers* here for a duration of $t_j^r - \bar{\tau}_{j_i}^r$, and then continues hovering during the video capture. If a drone arrives at $\lambda_{j_i}^r$ after t_j^r, it is invalid since the video capture for the activity cannot be conducted for the whole duration. So, $\bar{\tau}_{j_i}^r \leq t_{j_i}^r \leq \tau_{j_i}^r$. Also, since the deadline for on-time computation over the captured data is $\delta_{j_i}^r$, we require $\delta_{j_i}^r \geq \bar{t}_{j_i}^r$ in the activity specification. Once the drone finishes capturing video for the last activity in its r^{th} trip, it returns back to the depot at time $\bar{\tau}_{j_{n+1}}^r = \tau_{j_n}^r + \mathcal{F}(\lambda_{j_n}^r, \widehat{\lambda})$. So, the *total flying time* for a drone d_j for its r^{th} trip is:

$$f_j^r = \sum_{i=0}^{n} (\bar{\tau}_{j_{i+1}}^r - \tau_{j_i}^r)$$

and the *total hover time* for the drone on that trip is:

$$h_j^r = \sum_{i=1}^{n} (t_{j_i}^r - \bar{\tau}_{j_i}^r) + \sum_{i=1}^{n} (\bar{t}_{j_i}^r - t_{j_i}^r) = \sum_{i=1}^{n} (\bar{t}_{j_i}^r - \bar{\tau}_{j_i}^r)$$

which includes hovering due to early arrival at a waypoint, and hovering during the data capture. Once the video is captured for an activity, it can subsequently be processed on-board the drone. Let the mission scheduling algorithm assign the *time slot* $[\theta_{j_i}^k, \bar{\theta}_{j_i}^k)$ for executing a batch $b_{j_i}^k$ of activity α_{j_i} on drone d_j, where $\bar{\theta}_{j_i}^k = \theta_{j_i}^k + \rho_i^k$, based on the batch execution time. We define *completion functions* for processing each activity α_{j_i}, given by:

- The *data capture completion* function $u_{j_i} \in \{0, 1\}$ has a value of 1 if the drone hovers at location λ_{j_i} for the entire period from t_{j_i} to $\bar{t}_{j_i}$, and is 0 otherwise.
- The *on-board completion* function $0 \leq \bar{u}_{j_i} \leq 1$ indicates the fraction of batches of that activity that are completed on-board the drone. Let $\bar{\mu}_i^k = 1$ if the batch b_i^k of activity α_i is completed on-board, and $\bar{\mu}_i^k = 0$ if it is not completed on-board the drone. Then, $\bar{u}_{j_i} = \frac{\sum_k \bar{\mu}_i^k}{q_i}$.

- The *on-time completion* function $0 \leq \bar{\bar{u}}_{j_i} \leq 1$ gives the fraction of batches of that activity that are fully completed within the deadline. As before, let $\bar{\bar{\mu}}_i^k = 1$ if the batch b_i^k of activity α_i is completed on-time, i.e., $\bar{\theta}_i^k \leq \delta_i$, and $\bar{\bar{\mu}}_i^k = 0$ otherwise. So, $\bar{\bar{u}}_{j_i} = \frac{\sum_k \bar{\bar{\mu}}_i^k}{q_i}$.

Given these, the *total utility* for an activity α_i is given by: $U_i = u_i\gamma_i + \bar{u}_i\bar{\gamma}_i + \bar{\bar{u}}_i\bar{\bar{\gamma}}_i$, and the *total computation time* of batches on a drone d_j is:

$$c_j = \sum_{\alpha_i \in A} (\bar{\mu}_{j_i}^k + \bar{\bar{\mu}}_{j_i}^k) \cdot \rho_i^k$$

B. Optimization Objective of MSP

Based on these, the **objective** of the optimization is: $\arg\max \sum_{\alpha_i \in A} U_i$, i.e., assign activity waypoints along drone trips and routes, and assign activity batches to the drones' computing slots, to maximize the utility from data capture, on-board computation and on-time computation. These are subject to the following constraints on the execution slot assignments for a batch on a drone:

$$(t_{j_i} + k \cdot \beta) \leq \theta_{j_i}^k \qquad \bar{\theta}_{j_i}^k \leq \theta_{j_i}^{k+1} \qquad \bar{\theta}_i^k \leq \bar{\tau}_{j_{n+1}}$$

i.e., the data capture for a duration of β for the k^{th} batch of the activity is completed before the execution slot of the batch starts; the batches for each activity are strictly executed in sequence; and the execution of a batch should complete before the drone lands at the end of its mission. Also, there can only be one batch executing at a time on a drone. So $\forall[\theta_{j_p}^x, \bar{\theta}_{j_p}^x)$ and $[\theta_{j_q}^y, \bar{\theta}_{j_q}^y)$ slots assigned to batches b_p^x and b_q^y on drone d_j, we have $[\theta_{j_p}^x, \bar{\theta}_{j_p}^x) \cap [\theta_{j_q}^y, \bar{\theta}_{j_q}^y) = \varnothing$. Lastly, the *energy expended* by drone d_j on the r^{th} trip, to fly, hover, and compute, should be within its battery capacity:

$$E_j^r = f_j^r \epsilon^f + h_j^r \epsilon^h + c_j^r \epsilon^c \leq E$$

Table I lists the notations used in this paper.

V. Optimal Solution for MSP

In this section, we prove that MSP is NP-hard, and we define an optimal, but computationally slow, algorithm called Optimal Mission Scheduler (OPT) based on MILP.

A. NP-Hardness of MSP

The MSP combines elements of VRP/JSP in assigning routes and batches to drones. So, we are ready to prove that:

Theorem 1: MSP is NP-hard.

Proof: VRP is NP-hard [23]. Also, MSP considers multiple-trips, time-windows, energy-constraints, and utilities.

The VRP with multiple-trips (MTVRP), which considers a maximum travel time horizon T_h, is NP-hard. Any instance of VRP can be reduced in polynomial time to MTVRP by fixing the number of vehicles to equal the number of waypoints, $m = n$, and setting the time horizon $T_h = \sum_{e \in \mathcal{E}} \mathcal{F}(e)$, where $\mathcal{E}$ is the set of edges and $\mathcal{F}(e)$ is the flying time for traversing an edge [51], and limiting the number of trips to one. The VRP with time-windows (TWVRP), which limits the start and end time for visiting a vertex, $[t_i, \bar{t}_i)$, is NP-hard. Any instance of VRP can be reduced in polynomial time to TWVRP by just setting $t_i = 0$ and $\bar{t}_i = +\infty$ [15]. A VRP with energy-constrained vehicles is NP-hard, by just relaxing those constraints to match VRP.

TABLE I
List of Notations Used

Symbol	Description
$D = \{d_1, \ldots, d_m\}$	List of m drones d_j in fleet
$\widehat{\lambda} = (0, 0, 0)$	Location of depot at center of Cartesian space
π	Processing speed of drone (FLOPS)
E	Energy capacity of drone
$\epsilon^f, \epsilon^h, \epsilon^c$	Energy per unit time taken for flying, hovering, and computation
s	Constant speed at which the drone flies
$\mathcal{F}(\lambda_p, \lambda_q)$	Flying time between the locations λ_i and λ_j at speed s
$A = (\alpha_1, \ldots, \alpha_n)$	Set of n activities α_i to be scheduled
$\widehat{t} = 0$	Initial time at which activities are submitted
$\alpha_i = \langle \lambda_i, t_i, \bar{t}_i, \kappa_i, \delta_i, \gamma_i, \bar{\gamma}_i, \bar{\bar{\gamma}}_i \rangle$	Activity tuple
$\lambda_i = (x_i, y_i, z_i)$	Location of waypoint in Cartesian space where data is captured for activity α_i
$[t_i, \bar{t}_i)$	Time period of data capture for activity α_i
κ_i	Compute cost for processing the captured data (floating point operations) for activity α_i
δ_i	Deadline by which on-board processing should happen to get on-time utility for activity α_i
$\gamma_i, \bar{\gamma}_i, \bar{\bar{\gamma}}_i$	Utility benefit from data capture, on-board processing, and on-time processing for activity α_i
$B_i = (b_i^1, \ldots, b_i^{q_i})$	List of q_i batches that an activity's α_i's computation is divided into
β	Duration of captured data which forms a batch for processing
κ_i^k	Compute cost for processing a batch b_i^k
ρ_i^k	Processing time for a batch b_i^k
$[\theta_i^k, \bar{\theta}_i^k)$	Time slot assigned for executing batch b_i^k
$A_j = (\alpha_{j_1}, \ldots, \alpha_{j_n})$	List of activities assigned to a drone d_j's mission
$R_j = (\widehat{\lambda}, \lambda_{j_1}, \ldots, \lambda_{j_n}, \widehat{\lambda})$	List of waypoint locations in the mission route for a drone d_j
τ_{j_i}	Departure time for a drone d_j after data is captured for activity α_i at location λ_i
$\bar{\tau}_{j_{i+1}}$	Arrival time for a drone d_j at location λ_{i+1} for activity α_{i+1}, after flying from location λ_i
f_j	Total flying time for a drone d_j
h_j	Total hover time for a drone d_j
$u_i \in \{0, 1\}$	Data capture completion function for activity α_i
$\bar{\mu}_i^k \in \{0, 1\}$	On-board batch completion function of α_i
$\bar{\bar{\mu}}_i^k \in \{0, 1\}$	On-time batch completion function of α_i
$0.0 \leq \bar{u}_i \leq 1.0$	Fraction of batches of α_i that complete on-board but not on time
$0.0 \leq \bar{\bar{u}}_i \leq 1.0$	Fraction of batches of α_i that complete on-board and on-time
c_j	Total computation time of batches executing on a drone d_j

In the above VRP variants, the goal is only to minimize the costs. But MSP aims at maximizing the utility while bounding the energy and compute budget. In literature, the VRP variant with profits (PVRP) is NP-hard [26] since any instance of MTVRP can be reduced in polynomial time to PVRP by just setting all vertices to have the same unit-profit. Moreover, MSP has to deal with scheduling of batches for maximizing the profit. The original JSP is NP-hard [52]. So, any variant which introduces constraints is again NP-hard by a simple reduction, by relaxing those constraints, to JSP.

As MSP is a variant of VRP and JSP, it is NP-hard too. □

B. The OPT Algorithm

The Optimal Mission Scheduler (OPT) algorithm offers an optimal solution to MSP by modeling it as a multi-commodity flow problem (MCF), similar to

TABLE II
CONSTRAINTS THAT HAVE TO BE SATISFIED FOR THE OPT MILP FORMULATION

Expression	Description
$\sum_{k\in\mathcal{D}}\sum_{l\in\mathcal{R}}\sum_{j\in\overrightarrow{i}} x_{ij}^{kl} \leq 1, \quad \forall i \in \mathcal{V}'$	The waypoint for an activity α_i is visited only once.
$\sum_{j\in\overrightarrow{0}} x_{0j}^{kl} - \sum_{j\in\overleftarrow{0}} x_{j0}^{kl} = 0, \quad \forall k \in \mathcal{D}, l \in \mathcal{R}$	A drone trip l starting from the depot must also end there.
$\sum_{j\in\overrightarrow{0}} x_{0j}^{kl} = 1 \iff \sum_{j\in\overrightarrow{i}} x_{ij}^{kl} = 1, \quad \forall i \in \mathcal{V}', k \in \mathcal{D}, l \in \mathcal{R}$	A drone k must visit at least one waypoint on each trip l.
$\sum_{i\in\overleftarrow{j}} x_{ij}^{kl} - \sum_{i\in\overrightarrow{j}} x_{ji}^{kl} = 0, \quad \forall k \in \mathcal{D}, j \in \mathcal{V}', l \in \mathcal{R}$	A drone k visiting waypoint j must also fly out from there.
$(t_j - \mathcal{F}_{0j}) \cdot \sum_{k\in\mathcal{D}}\sum_{l\in\mathcal{R}} x_{0j}^{kl} \geq 0, \quad \forall j \in \mathcal{V}'$	Any drone flying to waypoint j from the depot must reach before its observation start time t_j.
$(t_j - \bar{t}_i - \mathcal{F}_{ij}) \cdot \sum_{k\in\mathcal{D}}\sum_{l\in\mathcal{R}} x_{ij}^{kl} \geq 0, \quad \forall i \in \mathcal{V}', j \in \overrightarrow{i}$	Any drone flying to waypoint j from i must reach before its observation start time t_j.
Let $\bar{\tau}_{k_{n+1}}^{l} = \sum_{i\in\mathcal{V}'} x_{i0}^{kl} \cdot (\bar{t}_i + \mathcal{F}_{i0}), \quad \forall k \in \mathcal{D}, l \in \mathcal{R}$. Then ... ... $\bar{\tau}_{k_{n+1}}^{l} \leq \tau_{\max}, \quad \forall k \in \mathcal{D}, l \in \mathcal{R}$	Defines the landing time of drone k at the depot after trip l. Depot landing times for all trips is within the maximum time.
$t_i + (g+1)\cdot\beta \leq \theta_i^g, \quad \forall i \in \mathcal{V}', g \in \mathcal{B}_i$	Batch g of activity α_i must be observed before it is processed.
$\bar{\theta}_i^g < \theta_i^{g+1}, \quad \forall i \in \mathcal{V}', g \in \mathcal{B}_i$	Processing of batch g of activity α_i must precede batch $g+1$.
$\sum_{j\in\overrightarrow{i}} x_{ij}^{kl} + \sum_{b\in\overrightarrow{a}} x_{ab}^{kl} - 1 \leq w_{ia}^{gh} + w_{ai}^{hg}, \quad \forall i,a \in \mathcal{V}', i < a, g \in \mathcal{B}_i, h \in \mathcal{B}_a, k \in \mathcal{D}, l \in \mathcal{R}$ $\bar{\theta}_i^g - \theta_a^h \leq M \cdot (1 - w_{ia}^{gh}), \quad \forall i,a \in \mathcal{V}, i \neq a, g \in \mathcal{B}_i, h \in \mathcal{B}_a$	Compute time slots of two batches g and h from activities α_i and α_a on the same drone k and trip l should not overlap [16].
$y_{ik}^{lg} = 1 \Rightarrow \bar{\theta}_i^g + M\left(1 - \sum_{j\in\overrightarrow{i}} x_{ij}^{kl}\right) \leq \delta_i, \quad \forall i \in \mathcal{V}', g \in \mathcal{B}_i, k \in \mathcal{D}, l \in \mathcal{R}$	Decision variable for batches that complete before deadline.
$z_{ik}^{lg} = 1 \Rightarrow \bar{\theta}_i^g + M\left(1 - \sum_{j\in\overrightarrow{i}} x_{ij}^{kl}\right) \leq \bar{\tau}_{k_{n+1}}^{l}, \forall i \in \mathcal{V}', g \in \mathcal{B}_i, k \in \mathcal{D}, l \in \mathcal{R}$	Decision variable for batches that complete before landing.
$\sum_{i\in\mathcal{V}}\left(\sum_{j\in\overrightarrow{i}}\left(x_{ij}^{kl}\cdot\mathcal{F}_{ij}\cdot\epsilon^f\right) + \sum_{g\in\mathcal{B}_i}\left(z_{ik}^{lg}\cdot\kappa_i^g\cdot\epsilon^c\right) + \sum_{j\in\overrightarrow{i}}\left(x_{ij}^{kl}\cdot(\bar{t}_j - (\bar{t}_i + \mathcal{F}_{ij}))\cdot\epsilon^h\right)\right) \leq E, \quad \forall k \in \mathcal{D}, l \in \mathcal{R}$	Sum of energy consumed for flying, hovering and computing on trip l of drone k should be within the battery capacity.

others [12], [53]. We reformulate the MSP definition as an MILP formulation.

The paths in the city are modeled as a *complete graph*, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, between the n waypoint vertices, $\mathcal{V} = \{0, 1, \ldots, n\}$, where 0 is the depot $\widehat{\lambda}$. Let $\overrightarrow{i}$ and $\overleftarrow{i}$ be the set of *out-edges* and *in-edges* of a vertex i, and $\mathcal{V}' = \mathcal{V} \setminus \{0\}$ be the set of all waypoints. We enumerate the m drones as $\mathcal{D} = \{1, \ldots, m\}$. Let $\tau_{\max}$ be the maximum time for completing all the missions, and $r_{\max}$ the maximum trips a drone can perform. Let $\mathcal{R} = \{1, \ldots, r_{\max}\}$ be the set of all possible trips. Let $x_{ij}^{kl} \in \{0, 1\}$ be a decision variable that equals 1 if the drone $k \in \mathcal{D}$ in its trip $l \in \mathcal{R}$ traverses the edge (i, j), and 0 otherwise. If $x_{ij}^{kl} = 1$ for $i \in \mathcal{V}'$, then the waypoint for activity α_i was visited by drone k on trip l. Let $\mathcal{B}_i = \{0, \ldots, q_i\}$ be the set of batches of an activity α_i. Let w_{ia}^{gh} be a binary decision variable used to linearize the batch computation whose value is 1 if batch b_i^g is processed before b_a^h, and 0 otherwise [16]. Let y_{ig}^{kl} be a decision variable that equals 1 if the drone $k \in \mathcal{D}$ in trip $l \in \mathcal{R}$ processes the batch g of activity α_i within its deadline δ_i, and 0 otherwise; and similarly, z_{ig}^{kl} equals 1 if the batch is processed before the drone completes the trip and lands, and 0 otherwise. Let the per-batch utility for data capture be $\Gamma_i = \frac{\gamma_i}{q_i}$, the on-board completion be $\bar{\Gamma}_i = \frac{\bar{\gamma}_i}{q_i}$, and the on-time completion be $\bar{\bar{\Gamma}}_i = \frac{\bar{\bar{\gamma}}_i}{q_i}$, for activity α_i. Let M be a sufficiently large constant. Given these, the MILP objective is to maximize the sum of the utility for data capture, on-board completion, and on-time completion of activities across multiple drones and multiple trips. Formally:

$$\max \sum_{k\in\mathcal{D}}\sum_{l\in\mathcal{R}}\sum_{i\in\mathcal{V}}\left(\sum_{j\in\overrightarrow{i}} x_{ij}^{kl}\cdot\Gamma_i\right) + \left(\sum_{g\in\mathcal{B}_i} y_{ig}^{kl}\cdot\bar{\Gamma}_i + z_{ig}^{kl}\cdot\bar{\bar{\Gamma}}_i\right) \tag{1}$$

subject to the constraints listed in Table II.

VI. HEURISTIC ALGORITHMS FOR MSP

Since MSP is NP-hard, OPT is not tractable for large inputs, time-efficient algorithms are necessary. In this section, we propose five heuristics, called *Cluster and Schedule* (CS), *Route, Split, and Schedule* (RSS), *Highest Utility First with Data capture* (HUFD), *Earliest Observation First with On-time processing* (EOFO), and *Interval Partition and Schedule* (IPS). Depending on the particular instance of the MSP to solve, some of them offer better utility and reduce the execution time while offering solutions with a lower utility.

A. The CS Algorithm

The *Cluster and Schedule* (CS) algorithm[2] aims to find near-optimal scheduling of batches while ignoring the optimizations of routing to conserve energy. CS is split into two phases: *clustering* and *scheduling*. Intuitively, in the *clustering* phase we group activities according to spatio-temporal parameters, to optimize the flight of drones). Then, in the *scheduling* phase, we assign these activities to the available drones. This is a *Divide and Conquer* approach.

Clustering Phase: First, we use the ST-DBSCAN algorithm [54] to find time-efficient spatio-temporal clusters of activities. It returns a set of clusters $\mathbb{C}$ such that for activities within a cluster $C_i \in \mathbb{C}$, certain spatial and temporal distance thresholds are met. Drones are then allocated to clusters depending on their availability. This clustering tries to conserve the flight routes to proximate waypoints, and optimizes for flying time at a gross level.

For each C_i, let $T_i^U = \max_{\alpha_j\in C_i}(\bar{t}_j + \mathcal{F}(\lambda_j, \widehat{\lambda}))$ be the upper bound for the *latest landing time* for a drone servicing activities in C_i; also, let $T_i^L = \min_{\alpha_j\in C_i}(t_j - \mathcal{F}(\widehat{\lambda}, \lambda_j))$ be the lower bound for the *earliest take-off time*. Then, all the temporal windows $[T_i^L, T_i^U]$ for each $C_i \in \mathbb{C}$ are sorted with

[2]The CS heuristic was referred to as JSC in our earlier paper [18].

respect to T_i^L. Recalling that there are m drones available at $\widehat{t} = 0$, these are proportionally allocated to clusters depending on the current availability, which in turn depends on the temporal window. So, $c_1 = \frac{m}{n} \cdot |C_1|$ drones are allocated to cluster C_1 at time T_1^L and unassigned from the cluster at time T_1^U; $c_2 = \frac{m-c_1}{n} \cdot |C_2|$ drones allocated to cluster C_2 from time T_2^L to T_2^U (assuming $T_2^L < T_1^U$), and so on.

Scheduling Phase: Activities can be assigned only to available drones, i.e., those that have available time slots for the computation. The *feasibility* of assigning α_i to d_j, is tested by checking if the required flying and hovering energy is enough to visit $A_j \cup \alpha_i$; here, we ignore the batch processing energy. If feasible, the drone can update its take-off and landing times accordingly, and then schedule the subset of batches $\widehat{B_i} \subseteq B_i$ within the energy capacity. Assignments are done in two steps: *default* and *test and swap assignment*.

Default Assignment: For each $b_i^k \in \widehat{B_i}$, let $P_{b_i^k} = [t_k + i\beta, \delta_k)$ be the *preferred interval*; $Q_{b_i^k} \subseteq P_{b_i^k}$ be the *available preferred sub-intervals*, i.e., the set of periods where no other batch is scheduled; and $S_{b_i^k} = [\delta_k, \bar{\tau}_{j_{n+1}})$ be the *schedulable interval*, which exceeds the deadline but completes on-board. Clearly, $P_{b_i^k} \cap S_{b_i^k} = \varnothing$. The *default schedule* determines a suitable time slot for b_i^k. If $Q_{b_i^k} \neq \varnothing$, b_i^k is *first-fit* scheduled [55] within intervals of $Q_{b_i^k}$; else, if $Q_{b_i^k} = \varnothing$, the same first-fit policy is applied over intervals of $S_{b_i^k}$. If b_i^k cannot be scheduled even in $S_{b_i^k}$, it remains unscheduled.

Test and Swap Assignment: If *default assignment* has batches that *violate their deadline*, i.e., scheduled in S but not in P, we use the *test and swap assignment* to improve the schedule. Let $P_i^+ = \bigcup_i P_{b_i^k}$ be the union of the preferred intervals forming the *total preferred interval* for an activity α_i. Each batch b_i^k is tested for violating its deadline. If it violates, then batches b_j^h from other activities already scheduled in P_i^+ are identified and tested if they too violate their deadline. If so, b_j^h is moved to the next available slot in $S_{b_j^h}$, and its old time slot given to b_i^k. If b_j^h is in its *preferred interval* but has more slots available in this interval, then b_j^h is moved to another free slot in $P_{b_j^h}$ and b_i^k assigned to the slot that is freed. Else, the current configuration does not contain violations, except for the current batch b_i^k, but all available slots are occupied. So, the utility for b_i^k is compared with another b_j^h in P_i^+, and the batch with a higher utility gets this slot.

Algorithm 1 CS(A, D)

```
ℂ ← clustering phase;
for C_k ∈ ℂ do
    for α_i ∈ C_k do
        for d_j assigned to C_k do
            if α_i ∪ A_j is feasible then
                Apply best sched. from among default
                  and test and swap assignment on B̂_i;
```

The CS algorithm is listed in Algorithm 1. After the *clustering phase*, activities are tested for their feasibility. If so, the *default assignment* is evaluated in terms of total utility. If this creates deadline violation, the *test and swap assignment* performed, and the best scheduling is applied. The CS algorithm is guaranteed to terminate since we iteratively process each cluster and the number of clusters is finite (n in the worst case).

Time complexity of CS: ST-DBSCAN's time complexity is $\mathcal{O}(n \log n)$ for n waypoints [54]. Unlike k-means clustering, ST-DBSCAN automatically picks a suitable number of clusters, k, with $\approx \frac{n}{k}$ waypoints each. For k times, we compute the min-max of sets of size $\frac{n}{k}$, sort the k elements and finally make $\frac{n}{k}$ assignments. So this drones-to-clusters allocation takes $\mathcal{O}(k\frac{n}{k} + k \log k + \frac{n}{k})$ time. Hence, this *clustering phase* takes $\mathcal{O}(n \log n)$ time.

For the *test and swap assignment*, we maintain an interval tree for fast temporal operations. If l is the maximum number of batches to schedule per activity, building the tree costs $\mathcal{O}(\frac{nl}{k} \log(\frac{nl}{k}))$, while search, insertion and deletion cost $\mathcal{O}(\log(\frac{nl}{k}))$. Finding free time slots makes a pass over the batches in $\mathcal{O}(\frac{nl}{k})$. This is repeated for l batches, to give an overall time complexity of $\mathcal{O}(\frac{nl}{k} \log(\frac{nl}{k}) + \frac{n}{k} l^2)$. Also the *default assignment* relies on the same interval tree, reporting the same complexity as *test and swap assignment*.

Finally, for the k clusters and each application in a cluster, two schedule assignments are calculated for all the drones. Thus, the time complexity of CS, which is the sum of the time for the *clustering phase* and the *scheduling phase*, is $\mathcal{O}(n \log n) + \mathcal{O}(k\frac{n}{k} m(\frac{nl}{k} \log(\frac{nl}{k}) + \frac{n}{k} l^2))$. However, since the clustering can result in a single cluster, i.e., $k = 1$, and with $m \to n$, by substitution we obtain the *overall complexity* of CS as $\mathcal{O}(n \log n) + \mathcal{O}(n^2(nl \log(nl) + nl^2)) = \mathcal{O}(n^3 l^2)$ in the worst case.

B. The RSS Algorithm

The *Route, Split, and Schedule* (RSS) algorithm[3] aims to find near-optimal waypoint routing while initially ignoring efficient scheduling of the batch computation. RSS is split into three phases: *routing*, *splitting*, and *scheduling*. In the *routing* phase we create drones' routes within the temporal constraints of activities, while in the *splitting* phase we divide routes that are energy unfeasible into smaller feasible sub-routes. Then, in the *scheduling* phase, we suitably schedule the batches on each drone. This follows a *Greedy* approach.

Routing phase: In this phase, RSS builds routes while satisfying the *temporal constraint* for activities, i.e., for any two consecutive activities (α_i, α_{i+1}) in the route, $\bar{t}_i + \mathcal{F}(\lambda_i, \lambda_{i+1}) \leq t_{i+1}$. This is done using a modified version of k-nearest neighbors (k-NN) algorithm, whose solution is then locally optimized using the 2-OPT* heuristic [56], as follows. Starting from $\widehat{\lambda}$, a route is iteratively built by selecting, from among the k nearest waypoints which meet the *temporal constraint*, the one, say, λ_1 whose activity has the earliest *observation start time*. This process resumes from λ_1 to find λ_2, and so on until there is no feasible neighbor. $\widehat{\lambda}$ is finally added to conclude the route. This procedure is repeated to find other routes until all the possible waypoints are chosen. This initial set of routes is optimized to minimize the flying and hovering energy using 2-OPT*, which lets us find a local optimal solution from the given one [15]. However, routes found here may be infeasible for a drone to complete within its energy constraints.

[3]The RSS heuristic was referred to as VRC in our earlier paper [18].

Splitting phase: Say $R_{i,j} = (\widehat{\lambda}, \lambda_i, \ldots, \lambda_j, \widehat{\lambda})$ be an energy-infeasible route from the routing phase, which visits λ_i/λ_j as the first/last waypoints from/to $\widehat{\lambda}$. The goal is to find a suitable waypoint λ_g for $i \leq g < j$ such that by splitting $R_{i,j}$ between λ_g and λ_{g+1}, we can find an energy-feasible route while also improving the overall utility and reducing scheduling conflicts for batches. For each edge $(\lambda_g, \lambda_{g+1})$, we compute a *split score* whose value sums up three components: *energy score*, *utility score*, and *compute score*.

Energy score: Let $E(a, b)$ be the cumulative flying and hovering energy required for some route $R_{a,b} \subseteq R_{i,j}$. Here we sequentially partition the route $R_{i,j}$ into multiple *viable trips* $R_{(i,k_1-1)}$, $R_{(k_1,k_2-1)}$, …, $R_{(k_x,j)}$ such that each is a maximal trip while also remaining energy-feasible, i.e., $E(k_y, k_{y+1} - 1) \leq E$ while $E(k_y, k_{y+1}) > E$. For each $(\lambda_g, \lambda_{g+1}) \in R_{(k_y,k_{y+1}-1)}$, the *energy score* is the ratio $\frac{E(k_y,g)}{E} \leq 1$. A high value indicates that a split at this edge reduces the energy usage.

Utility score: Let $U(a, b)$ give the cumulative data capture utility from visiting waypoints in a route $R_{a,b} \subseteq R_{i,j}$. Say edge $(\lambda_g, \lambda_{g+1}) \in R_{(k_y,k_{y+1}-1)} \subseteq R_{i,j}$ is also part of a viable trip from above. Here, we find the data capture utility of a sub-route of $R_{i,j}$ that starts a new maximal viable trip at λ_{g+1} and spans until λ_l as $U(g, l)$. The utility score of edge $(\lambda_g, \lambda_{g+1})$ is the ratio between this new maximal viable trip and the original viable trip the edge was part of, $\frac{U(g,l)}{U(k_y,k_{y+1}-1)}$. A value > 1 indicates that a split at this edge improves the utility relative to the earlier sequential partitioning of the route.

Compute score: We first do a soft-scheduling of the batches of all waypoints in $R_{i,j}$ using the *first-fit* scheduling policy, mapping them to their *preferred interval*, which is assumed to be free. Say there are $|R_{i,j}|$ such batches. Then, for each edge edge $(\lambda_g, \lambda_{g+1}) \in R_{i,j}$, we find the overlap count O_g as the number of batches from α_g whose execution slot overlaps with batches from all other activities. The overlap score for edge $(\lambda_g, \lambda_{g+1})$ is given as $\frac{O_g}{|R_{i,j}|}$. If this value is higher, splitting the route at this point will avoid batches from having schedule conflicts in their preferred time slot.

Once the three scores are calculated and summed up, the edge with the highest *split score* is selected as the split-point to divide the route into two sub-routes. If a sub-route meets the energy constraint, it is selected as *valid trip*. If either or both of the sub-routes exceed the energy capacity, the splitting phase is recursively applied to the sub-route(s) till all waypoints in the original route are part of valid trips.

Scheduling phase: Trips are then sorted in decreasing order of their total utility, and drones are allocated to trips depending their temporal availability. Once assigned to a trip, the drone's scheduling is done by comparing the *default assignment* and the *test and swap assignment* used in CS.

The RSS algorithm is given in Algorithm 2. After the *routing phase*, the energy-unfeasible routes are split into feasible ones in the *splitting phase*, and then drones are allocated to them. Finally, the *scheduling phase* is applied to find the best schedule between the *default assignment* and the *test and swap assignment*. The RSS algorithm will always terminate since we iteratively assign routes to drones. The number of routes and drones is finite and hence the iteration will terminate.

Time complexity of RSS: In the *routing phase*, the modified k-NN with n waypoints and k number of neighbors takes $\mathcal{O}(kn)$. 2-OPT* takes $\mathcal{O}(n^4)$ in time [57]. Hence, this phase overall has a cost of $\mathcal{O}(n^4)$.

Algorithm 2 RSS(A, D)

```
ℝ ← routing phase;
for R_ij ∈ ℝ do
    for (λ_g, λ_{g+1}) ∈ R_ij, i ≤ g < j do
        s(g) ← energy score + utility score + compute score;
ℝ' ← splitting phase based on scores s(i), 1 ≤ i ≤ n;
for d_j assigned to R_ij ∈ ℝ' do
    Apply best scheduling from among default assignment and test and swap assignment on R_ij;
```

In the *splitting phase*, calculating the energy score for a route with length n edges takes $\mathcal{O}(n)$. Calculating the energy score has $\mathcal{O}(n^2)$ complexity, and calculating the compute score has $\mathcal{O}(n)$ complexity. Considering a recursion of length $n - 1$, the complexity of this phase is $\mathcal{O}(n^3)$

Combining *default assignment* and *test and swap assignment*, the RSS's *overall complexity* is $\mathcal{O}(n^4) + \mathcal{O}(n^3) = \mathcal{O}(n^4)$ in the worst case.

C. The HUFD Algorithm

The *Highest Utility First with Data capture* (HUFD) algorithm aims to greedily assign the activities with the largest utility to drones while initially ignoring other optimizations related to the routing of drones and the scheduling of batches. HUFD is split into three phases: *sorting*, *routing*, and *scheduling*. The *sorting* phase orders the activities by their maximum possible utility, and in the *routing* phase we assign activities to drones. Finally, in the *scheduling* phase, we schedule the activities on each drone. This takes a *Greedy* approach.

Sorting phase: Here, HUFD initially sorts all the activities $\alpha_i \in A$ in non-decreasing order of the total potential utility and creates the ordered set $\mathbb{A}$. Let $\widehat{U_i} = \gamma_i + \bar{\gamma}_i + \bar{\bar{\gamma}}_i$ be the *potential utility* for an activity α_i, i.e., the sum of data capture, on-board, and on-time utilities. After the sorting procedure we have $\widehat{U_i} \geq \widehat{U_{i+1}}$ for each activity in $\mathbb{A}$.

Routing phase: We now start to assign the sorted activities to drones. Each drone starts with an empty trip, $(\widehat{\lambda}, \widehat{\lambda})$. For each (remaining) activity $\alpha_i \in \mathbb{A}$ in sorted order of total utility, we iterate through all the drones and all their trips to identify to which trip this activity can be added to while meeting the energy constraints of the trip, i.e., we do not exceed the energy capacity to go from the depot, to all the existing activities of the trip and also the new activity, and back to the depot. For each such trip, we also calculate the increase in the energy usage for that trip as a result of adding this activity. From among all the energy-feasible trips, we identify the one with the least increase in energy as the one to add this activity to.

If none of the existing trips can accommodate this activity without exceeding their energy budget, we also check if the activity can be placed solo in a new trip on a drone – from the depot to the new activity location and back to the depot. Here, besides the energy constraint being met, the activity deadline of the trip should also be feasible given the other trips already scheduled on the drone. If so, this will create a new trip on a drone. As an optimization, we only need to check empty

trips once for their feasibility to add the activity irrespective which drone such a trip is present in. If any of the trips can be assigned this activity, we are guaranteed to accrue its data-capture utility; if not, this activity is dropped. In either case, we remove the activity from $\mathbb{A}$ and repeat this process for the next activity with the highest potential utility.

Scheduling phase: When a new activity α_i is added to the current set of activities A_j for a drone d_j, HUFD tries an optimistic scheduling by checking if all its batches can be scheduled without deadline violations using the *default assignment* of CS algorithm (see Section VI-A) for the new batches. This assignment is optimistic; in case α_i introduces violations, HUFD does not reschedule the batches. So, the drone d_j may not necessarily obtain the total potential utility $\widehat{U_i}$ of the activity α_i since the full activity may not be successfully scheduled for processing on-time or even on-board the drone.

Algorithm 3 HUFD(A, D)

```
1 𝔸 ← sort A by non-descending order of total potential utility;
2 for α_i ∈ 𝔸 do
3     d_k ← ∅;
4     for d_j ∈ D do
5         if α_i ∪ A_j is feasible for energy capacity of d_j and d_j has lower incremental energy use than d_k then
6             d_k ← d_j;
7     Assign α_i to d_k if d_k ≠ ∅ and try to schedule its batches using default assignment;
```

The HUFD algorithm is shown in Algorithm 3. The main idea is to prioritize assignment of the activities with highest potential utility to the drones. After the initial *sorting phase* where activities are ordered by the total potential utility from high to low, activities are iteratively picked for *routing* on any drone d_k which has the lowest feasible energy cost to complete the activity's data capture. Then, the activity is scheduled for processing on this drone d_k using *default assignment* and if there are deadline violations for its batches, those are left unscheduled. The HUFD algorithm will terminate after we iteratively process each activity and assign it to drones. The iteration never reconsiders an activity and hence it is bounded.

Time complexity of HUFD: The *sorting phase* takes $\mathcal{O}(n \log n)$ for sorting n activities. In the *routing phase* the activity with the largest total utility is selected and attempted to be assigned to a drone. Evaluating the route with the minimum total energy cost for visiting $A_j \cup \alpha_i$, given that A_j is already energy-feasible, is computationally inexpensive because the data capture time windows of the activities limit the possible choices. Since there are m drones, and that the *scheduling phase* by invoking *default assignment* takes $\mathcal{O}(nl^2)$ (see Section VI-A), the loop costs $\mathcal{O}(mnl^2)$.

So, the *overall complexity*, which depends on the *sorting phase* and on the *routing phase*, is $\mathcal{O}(n \log n) + \mathcal{O}(mnl^2)$ in the worst case. If $m \ll n$, then m can be treated as a constant and the time complexity of HUFD is $\mathcal{O}(n \log n + nl^2) = \mathcal{O}(n \log n)$. Otherwise, if $m \approx n$, then the time complexity is $\mathcal{O}(n \log n + n^2 l^2) = \mathcal{O}(n^2 l^2)$ because n^2 dominates $n \log n$.

D. The EOFO Algorithm

The *Earliest Observation First with On-time processing* (EOFO) algorithm assigns activities with the earliest observation start time to drones, and further also avoids deadline violations for processing batches. Accordingly, when activities are assigned to drones, the data capture, on-board, and on-time utilities are fully obtained in this algorithm. HUFD is split into two phases: *sorting* and *assigning*. During *sorting*, we order the activities by their earliest observation time, and when *assigning*, we map the activities to drones while achieving on-time processing. This too is a *Greedy* approach.

Sorting phase: EOFO initially sorts all the activities $\alpha_i \in A$ in non-decreasing order of their earliest observation start time, creating the ordered set $\mathbb{A}$. Given that the *observation start time* for an activity α_i is t_i, after the sorting we will have $t_i \leq t_{i+1}$ for each activity in $\mathbb{A}$.

Assigning phase: The assignment phase combines the routing and scheduling phases of HUFD with the additional requirement that the batches of the activities being added to a route can be fully processed on-time. All drones start with an empty trip. EOFO greedily picks from $\mathbb{A}$ the next remaining activity with the earliest observation start time and looks for a drone d_j for it. For each drone, we check if adding α_i to the current list of activities A_j of any of its trips is feasible in terms of energy, and further, does not create any on-time batch processing deadline violation using just the *default assignment* (see Section VI-A). Among them, we select the drone whose trip requires the minimum increase in the energy consumed for the trip. If this activity cannot be placed in any of existing trips of the drones without energy or deadline violations, then we attempt to place it as a solo trip on a drone. If that too fails, the activity is dropped. We move on to assign the next activity in $\mathbb{A}$. EOFO will accrue the total potential utility for the activity in its entirety, compared to HUFD which will accrue the data capture utility but may not gain the processing utility if the activity's batches cannot be scheduled.

The EOFO algorithm is described in Algorithm 4. After the initial *sorting phase* is done based on the earliest observation start time of the activities, we iteratively picked each activity to *assign* it to any drones without on-time processing deadline violation, and choose the drone with the least increase in its total round trip energy on adding this activity. If none of the existing trips can accommodate this activity without exceeding their energy budget, we also check if the activity can be placed solo in a new trip on a drone - from the depot to the new activity location and back to the depot. If none of the drone meet these constraints, we leave the activity unassigned. The EOFO algorithm is guaranteed to terminate since we process each activity exactly once and assign it to drones.

Time complexity of EOFO: In the *routing phase*, the pre-processing sorting procedure takes $\mathcal{O}(n \log n)$. Since the activities are sorted by the earliest observation start time, it is possible to efficiently evaluate (in constant time) if the selected activity α_i causes violations of the on-time deadline in $A_j \cup \alpha_i$. As for HUFD, the *assigning phase* takes $\mathcal{O}(mnl^2)$. Accordingly, the EOFO's *overall complexity* is $\mathcal{O}(n \log n + mnl^2)$ in the worst case, similar to HUFD.

E. The IPS Algorithm

The *Interval Partition and Schedule* (IPS) algorithm creates round-trip routes of waypoints such that there is no violation

Algorithm 4 EOFO(A, D)

$\mathbb{A} \leftarrow$ *sort* A *by non-desc. order of earliest obs. start time*;
for $\alpha_i \in \mathbb{A}$ **do**
  $d_k \leftarrow \varnothing$;
  for $d_j \in D$ **do**
    if $\alpha_i \cup A_j$ *is feasible for energy capacity of* d_j ***and*** *on-time compute scheduling on* d_j ***and*** *if* d_j *has lower incremental energy use than* d_k **then**
      $d_k \leftarrow d_j$;
  Assign α_i to d_k if $d_k \neq \varnothing$ and schedule its batches using *default assignment*;

of the on-time processing deadline. Each route is then seen as a temporal interval, and each drone is assigned a set of non-overlapping intervals where each route corresponds to a new trip for the drone. IPS is split into two phases: *interval creation* and *scheduling*. In the *interval creation* phase we identify feasible routes while considering the activity deadlines, and in the *scheduling* phase we actually assign drones to these routes. This is a *Divide and Conquer* approach.

Interval partitioning phase: Here, the objective is to create a set of feasible routes $\mathbb{R}$ for the drones from/to the depot. In principle, since there are n activities, the number of possible routes is exponential, 2^n, and hence it is intractable to evaluate all of them. However, in practice the actual number of routes is much fewer due to the presence of activity deadlines and energy requirements to be met by the drones. Initially, all the activities α_i are sorted in non-decreasing order of their earliest observation start time to give the ordered set $\mathbb{A}$ such that $t_i \leq t_{i+1}$. Then, we create the set of all energy-feasible contiguous routes $\mathbb{R}$ formed from $R_{i,j} = (\widehat{\lambda}, \lambda_i, \lambda_{i+1}, \ldots, \lambda_j, \widehat{\lambda})$, where λ_i is the waypoint associated to the activity α_i and for all $i = 1, \ldots, n$ and $i \leq j \leq n$. The route $R_{i,j}$ is added to the set $\mathbb{R}$ if it can be completed within the energy budget of a single trip for a drone. So, starting with $i = 1$, the first route $R_{1,1}$ only has the activity $\alpha_1 \in \mathbb{A}$, the next route $R_{1,2}$ has α_1 and α_2, and so on. Then, we move to $i = 2$ and evaluate the route $R_{2,2}$ with activity α_2 and so on. All routes $R_{1,1}, R_{1,2}, \ldots, R_{2,2}, \ldots$, including the *singleton* routes $R_{i,i}$, that are also energy-feasible are present in $\mathbb{R}$. At the end, the maximum number of possible trips is $n(n+1)/2 \approx \mathcal{O}(n^2)$.

Scheduling phase: For each route $R_{i,j} \in \mathbb{R}$ we have its corresponding overall potential utility summed across all its activities, i.e., $\widehat{U}_{i,j} = \widehat{U}_i + \ldots + \widehat{U}_j$. Now, these need to be assigned to drones while avoiding violations of the on-time processing deadline and ensuring that each activity is assigned to no more than one drone.

Let $T^T_{i,j} = t_i - \mathcal{F}(\widehat{\lambda}, \lambda_i)$ be the take-off time required for route $R_{i,j} \in \mathbb{R}$, and $T^L_{i,j} = \bar{t}_j + \mathcal{F}(\lambda_j, \widehat{\lambda})$ be its landing time. Hence, we can treat each route as a *temporal interval* delimited by its take-off and landing time, $(T^T_{i,j}, T^L_{i,j})$. Given two routes $R_{a,b}$ and $R_{c,d}$, they are said to be *compatible for the same drone* if $T^L_{a,b} \leq T^T_{c,d}$ or $T^L_{c,d} \leq T^T_{a,b}$, i.e., it is possible for a single drone to service these two routes on two different trips during this mission.

We can model these temporal interval for the routes as an undirected graph $\mathcal{G}$ whose vertices are the routes $R_{i,j}$, and an edge exists between any pair of vertices if the two routes are *not compatible* for being assigned to the same drone. The weight of a vertex is the overall potential utility $\widehat{U}_{i,j}$ for the route. Now, the goal is to find the *Weighted Maximal Independent Sets (WMIS)* of vertices from this graph. A WMIS identifies a set of vertices in the graph that do not have any edge between them and the sum of their vertex weights is the highest among all such sets. The WMIS indicates a set of routes whose intervals are *compatible* for a single drone's mission and will maximize the overall potential utility. Solving WMIS is NP-hard for general graphs. But the graph $\mathcal{G}$ is an *interval graph*, which is a type of *chordal graph*, and WMIS can be efficiently solved in polynomial time for chordal graphs [58], [59].

In the scheduling phase, we iteratively solve WMIS m times, once for each drone. Specifically, it is first invoked on the entire graph $\mathcal{G}$ and returns a set of compatible energy-feasible trips (vertices) for the first drone. Then we remove these vertices from the $\mathcal{G}$, along with trips (vertices) whose activities are already assigned to the first drone. Then we repeat WMIS on the updated graph and the resulting trips are assigned to the next drone, and so on until either all valid vertex sets are assigned or we run out of drones. The actual scheduling of the batches of an activity for on-board processing is again done optimistically using the *default assignment*. In case of deadline violations, we do not reschedule the batches. So, we are guaranteed to accrue the data capture utility but not the on-board or on-time utility for an activity.

Algorithm 5 IPS(A, D)

$\mathbb{A} \leftarrow$ *sort* A *by non-desc. order of earliest obs. start time*;
$\mathbb{R} \leftarrow$ *energy-feasible routes from* $\mathbb{A}$;
$\mathcal{G} \leftarrow$ *form interval graph from* $\mathbb{R}$;
for $d_j \in D$ **do**
  Assign activities for a route-set from *WMIS(*$\mathcal{G}$*)* to d_j;
  Try to schedule the compute for the route-set's activities on d_j using *default assignment*;
  Remove the assigned route-set vertices from $\mathcal{G}$;

Algorithm 5 gives the pseudo-code for IPS. In the initial *intervals creation phase* activities are sorted by their earliest observation start time. Then, the set of energy-feasible trips is created by iterating over the sorted activities. These trips form time-intervals vertices in an undirected interval graph with vertex weights indicating the potential utility of the trip and edges between incompatible vertices. Finally, in the *scheduling phase*, we iteratively solve the WMIS problem on the graph to return the best compatible trip to maximize total potential utility for a drone and prune the interval vertices from the graph. The IPS algorithm is guaranteed to terminate since we iterate over each drone exactly once to assign them activities.

Time complexity of IPS: In the *intervals creation phase*, activities are sorted taking $\mathcal{O}(n \log n)$ time. Then, energy-feasible trips are created taking $\mathcal{O}(n^2)$ because the maximum number of possible trips is upper-bounded by $n(n+1)/2 \approx \mathcal{O}(n^2)$. Given $\hat{n}$ as the total length of the interval, WMIS can be optimally solved in $\mathcal{O}(\hat{n} \log \hat{n})$ time [60]. Hence, for solving the WMIS m times starting from the initial set of

TABLE III
COMPARISON BETWEEN THE ALGORITHMS

Algorithm	*Time Complexity*
CS	$\mathcal{O}(n^3l^2)$
Approach: Create spatio-temporal clusters, proportionally assign drone(s) to form energy-feasible trips. Schedule batches of a trip, try to meet on-time deadline.	
RSS	$\mathcal{O}(n^4)$
Approach: Create temporally feasible routes. Split into energy-feasible trips that reduce batch overlaps/improve utility. Schedule trips to drones, batches within a trip. Try to meet on-time deadline.	
HUFD	$\mathcal{O}(n \log n + mnl^2)$
Approach: Sort activities by highest potential utility. Create energy-feasible trips. Optimistically schedule the batches of a trip.	
EOFO	$\mathcal{O}(n \log n + mnl^2)$
Approach: Sort activities by earliest observation start time. Create energy-feasible trips that also have on-time batch completion.	
IPS	$\mathcal{O}(mn^2 \log n + ml)$
Approach: WMIS finds temporally compatible trips for each drone with high potential utility. Optimistically schedule the batches.	

trips with $\hat{n} = \mathcal{O}(n^2)$ intervals, takes $\mathcal{O}(m(n^2 \log n^2))$ time. The time complexity for assigning l batches using default schedule is $\mathcal{O}(l)$ So, the *overall time complexity* of IPS is $\mathcal{O}(n \log n + m(n^2 \log n^2) + ml) = \mathcal{O}(mn^2 \log n + ml)$.

Table III compares our proposed algorithms for MSP.

VII. PERFORMANCE EVALUATION

In this section, we report the results and analysis of our detailed experiments to evaluate the 5 heuristics and the optimal solution using realistic drone benchmarks and traces.

A. Experimental Setup

The OPT algorithm is implemented using IBM's CPLEX MILP solver v12 [61] with Python used to wrap the objective and constraints, and invoke the parallel solver. This gives the optimal solution as a best-case baseline. The CS, RSS, HUFD, EOFO, and IPS heuristics have a sequential implementation in native Python. By default, the heuristics run on server with Intel Xeon Gold 6208U CPU with 16 cores and 32 hyper-threads at 2.9 GHz, and 128 GB RAM. OPT uses all 32 threads while our heuristics run on 1 thread.

We perform real-world benchmarks on flying, hovering, DNN inferencing, and endurance, on a custom, commercial-grade drone. The X-wing quad-copter is designed with a top speed of 6 m/s (20 km/h), $<$ 120 m altitude, a 24000 mAh Li-Ion battery, and a payload capacity of 3 kg. It includes dual (front and downward) HD cameras, GPS and LiDAR Lite, and uses the Pixhawk2 flight controller. It also has an NVIDIA Jetson TX2 compute module with 4-Core ARM64 CPU, 256-core Pascal CUDA cores, 8 GB RAM, and 32 GB eMMC storage. The maximum flying time is $\approx$ 30 min with a range of 3.5 km. Based on our benchmarks, we use the following parameters in our experiments.

s	ϵ^f	ϵ^h	ϵ^c	E
4 m/s	750 J/s	700 J/s	20 J/s	1350 kJ

B. Workloads

We evaluate the scheduling algorithms for two *waypoint placement scenarios*: *Random* (RND) and *Depth First Search* (DFS), and several *activity configurations* to form 11 *workloads*. We use a subset of the road network for Bangalore, India extracted from Open Street Maps [62]. The sub-network has 481 vertices and 1402 edges of primary roads and highways, covering a radius of 3.5 km, with the depot being located at its center. All workloads have a maximum mission time (*makespan*) of 4 h spanning multiple trips. In the *RND placement*, n waypoints are randomly placed within a 3.5 km radius from the depot, and with a randomly generated activity's start time in $(0, 240]$ mins. This is an adversarial scenario with no spatio-temporal locality. The *DFS placement* is motivated by realistic traffic monitoring needs. We perform a depth-first traversal over the road network starting from the depot. For each hop in the DFS traversal, we choose a vertex as an activity waypoint with a probability of $p = 0.1$. For each hop in the traversal where we do not pick a vertex, we increase the probability for picking the next hop vertex by $p = p + 0.1$; if chosen, the probability is reset to $p = 0.1$. The *observation start time* of the activities grows monotonically: $t_i = \bar{t}_{i-1} + p \cdot 3000$ s, where t_i is the start time (in secs) of the i^{th} activity and $\bar{t}_{i-1}$ is the observation end time of the previous vertex picked in the DFS. Clustering of activities in CS is done using a Python implementation of ST-DBSCAN. We set the temporal threshold as 5 $mins$ and spatial threshold as 1 km to determine which points are placed in the same cluster.

$\bar{t} - t$	β	ρ_M	ρ_R	δ	$\gamma, \bar{\gamma}, \bar{\bar{\gamma}}$	(m, x)	n
$[1, 5]$ min	60 s	11 s	98 s	120 s	$[1, 5]$	$(5, 20)(10, 10)(20, 5)$	100

The first activity workload uses the DFS and RND waypoint placement coupled with activity and drone configurations shown in the table above. These parameters are based on reasonable operational assumptions and schedule feasibility. The compute tasks for the activity are *SSD Mobilenet v2 DNN* (MNet) [63], popular for analyzing drone footage [64], and *FCN Resnet18 DNN* (RNet) [65] on the TX2. The activity instances in the workload have a fixed batching interval (β), batch execution time (ρ_M for MNet, ρ_R for RNet), and deadline (δ). We uniformly choose the data capture time ($\bar{t} - t$) and values for the three utility ($\gamma, \bar{\gamma}, \bar{\bar{\gamma}}$) from the range specified in the table. We use a fixed number of activities ($n = 100$), but vary the number of drones (m) and the *load factor* x, which decides the maximum activities per drone: $n = x \cdot m$. Drones make at most $r_{\max} = \frac{n}{m}$ trips. This gives us 625 possible configurations of activity instances to choose from for this workload, for a given number of drones and load factor.

We evaluate 3 *workload types*: MNet model with RND waypoint placement (RMN), MNet with DFS (DMN), and RNet with DFS (DRN); for brevity, RNet is only run on DFS. For each *instance* of a workload type, we sample $n = 100$ activity instances from these configurations. We ensure that all activities can be feasibly reached within its data capture start time by a drone starting from/returning to the depot.

We define an additional 7 *workload types* for RND placement, with more diverse activity processing times, deadlines and utilities. Here, we expand to five DNN/vision models: *MobileNet (M-Net)* [63], *Object detection on Drone data* [66], *Resnet-18 (R-Net)* [65], *Crowd counting using Drones* [67], and *Optical Flow* [68], with respective batch execution times of $\rho \in \{11, 37, 98, 124, 391\}$ s as measured on the TX2. An activity instance in a workload has equal probability of being assigned any of these model types. We retain the *data*

capture of $t - \bar{t} = [1, 5]$ min and batch execution interval of $\beta = 60$ s. Activities have different deadlines classified as *short-, medium-* and *long-term*. When assigning deadlines to activities, we ensure that the batch execution time does not exceed the sampled deadline. This implies that some models with high batch execution times may not feature in workload mixes of short deadlines. Their utilities also vary, as shown in the table below. For short-term, we assign a higher utility range for on-time processing ($\bar{\gamma}$) as it is time-critical and this range reduces for medium-term deadlines. The on-board processing utility ($\bar{\bar{\gamma}}$) is higher for medium and long-term deadlines, as these are opportunistic.

Activity Type	δ	γ	$\bar{\gamma}$	$\bar{\bar{\gamma}}$
short-term	$[60, 120]s$	$[1, 10]$	$[8, 10]$	0
medium-term	$[600, 1200]s$	$[1, 10]$	$[3, 7]$	$[8, 10]$
long-term	$[3600, 7200]s$	$[1, 10]$	1	$[8, 10]$

We form 7 workload types using different ratios of short, medium and long-term activity types for the n activities in a workload: $(1, 0, 0)$ for Short-term activities Only (RSO), $(0, 1, 0)$ for Medium-term Only (RMO), $(0, 0, 1)$ for Long-term Only (RLO), $(\frac{1}{2}, \frac{1}{4}, \frac{1}{4})$ for Short-term Dominant (RSD), $(\frac{1}{4}, \frac{1}{2}, \frac{1}{4})$ for Medium-term Dominant (RMD), $(\frac{1}{4}, \frac{1}{4}, \frac{1}{2})$ for Long-term Dominant (RLD), $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ for EQual weight (REQ). The short-term activities have $9,000$ distinct configurations to sample from for a workload instance; the medium-term have $450,000$ and long-term have $540,000$ possibilities. All activities chosen have feasible data capture times.

Last, we create a workload, *Prioritized Random (PRND)*, to demonstrate the effect of large skews in the activity utilities. This workload is identical to RMN on the activity placement, deadlines, and the observation start and end times. However, 5% of the activities have all three utility values set between $[7, 10]$ while the rest 95% have these utilities between $[1, 3]$.

C. Experimental Results

We run the algorithms to evaluate a schedule and the expected utility for each workload instance. For each of the 10 workload types, we sample different numbers of activities (n) and use different number of drones (m), and report results averaged over 10 workload instances of each type. While the initial results assume an ideal and deterministic performance of the drones, later in Section VII-C.3 we present results using a trace of the drone's actual flying behavior.

1) Comparison of OPT *With Heuristics for REQ:* Figure 2a reports the *average utility % that is achieved* out of the maximum theoretical utility that is achievable for a given workload, i.e., sum of the data capture and on-time processing utility for all its activities, for OPT and the 5 algorithms. Here, we *vary* the activity count $n = \{10, 20, 50, 100\}$ and the number of drones $m = 5, \ldots, 50$, but only evaluate the *REQ workload* which is the most generic of all. The bars are averaged over the 10 workload instances for each type, while the whiskers show the minimum and maximum utility % of the instances. The OPT is only run for 4 combinations of m and n due to the large amount of time per run – it takes > 1 h to solve for a single workload instance of $n = 20, m = 5$, and > 16 h to solve for one instance of $n = 20, m = 10$. While optimal, it is not useful to schedule our 4 h activity windows.

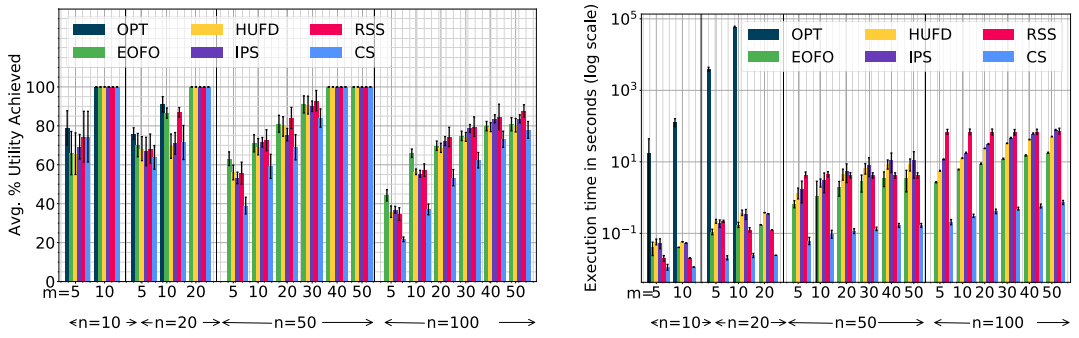
OPT *offers modest utility gains but is intractable for larger problems.* As evidenced by Figure 2a, when OPT (dark blue) finds a solution, it is better than all the heuristics by a modest $\approx 5\%$. OPT is viable with a small number of drones and activities, e.g., $n \leq 10$ and $m \leq 10$, where it executes within 10 mins. For larger instances, its required computation time blows up dramatically. So its marginal utility improvements are offset by the impractical time taken for larger problem sizes. Here, the heuristics are competitive. E.g., RSS and EOFO obtain only 4% less utility than OPT for $n = 20, m = 10$, and even in the worst case, HUFD has only 22% lower utility for $n = 20, m = 10$, but executes much faster. While it is possible to obtain 100% of the *data capture utility*, especially when $m = n$, the same cannot be said for the *on-time* and the *on-board processing utility* for the workload instances. Even OPT may not achieve 100% of the theoretical achievable utility due to deadline constraints on the computation and finite compute capacity.

OPT *is the slowest while* CS *is the fastest to solve.* Figure 2b plots the *execution time* in seconds (logarithmic scale) for the optimal algorithm OPT and the five heuristics, for the same scenario as Figure 2a. Despite OPT using $16\times$ more cores compared to the single-threaded execution of the heuristics, it is two orders of magnitude slower than HUFD for $m = 5$ and $n = 10$, which is the smallest workload we evaluate. This is consistent with the NP-hard nature of MSP. The relative ordering of the observed execution time of the algorithms is consistently CS (light blue) $<$ EOFO (green) $<$ HUFD (yellow) $<$ IPS (purple) $<$ OPT (dark blue), with RSS (red) being faster than EOFO or slower than IPS, depending on the conditions. Since RSS has a time complexity of $\mathcal{O}(n^4)$, its time taken rises faster than the other heuristics with increasing n. But it remains much faster than OPT, taking only 69 s even for $n = 100, m = 5$, keeping it tractable. As its complexity does not depend on m, its execution time remains constant across all m with a fixed n. Both HUFD and EOFO have the same time complexity of $\mathcal{O}(n \log n + mnl^2)$, and their algorithms are identical except for an extra condition check for EOFO. Despite that, the latter is slightly faster in practice.

CS is the fastest heuristic despite a high time complexity of $\mathcal{O}(n^3 l^2)$. In practice it executes quickly taking < 1 s for all our runs, indicating that the number of clusters formed are closer to m than n. E.g., for a large scenario with 5 drones and 100 activities, it takes only 0.3 s to return a schedule, which is $10\times$ faster than the next fastest algorithm EOFO which takes 2.7 s for the same scenario. However, the utility returned by CS is the worst, though it stays within 21% of the best heuristic solution on average. For this scenario, HUFD is $2\times$ slower than EOFO (5.6 s vs. 2.7 s), while IPS is $2\times$ slower than HUFD (5.6 s vs. 11.7 s).

The gains from adding more drones for a fixed number of activities is marginal. In Figure 2a, as the number of drones increases for a fixed number of activities, the improvement in utility % obtained diminishes and eventually saturates. E.g., for $n = 50$ activities, with $m = 5$ drones, the average utility is $(22, 34, 36, 44, 38)\%$ across the 5 heuristics, while quadrupling the number of drones to $m = 20$ gets us a utility % of only $(68, 83, 79, 81, 75)\%$, respectively. When the number of activities is equal to or almost equal to the number of drones, e.g., $m = 40, n = 50$, the average utility % obtained across all 5 heuristics is 100% since each drone can handle one or two independent activities successfully.

2) Utility for Varying Load Factors and Workloads: OPT is computationally intractable for larger numbers of activities and drones, and therefore here we compare only the heuristics. We estimate it would require more than a year to solve for



(a) % of total possible utility achieved.

(b) Algorithm execution time. OPT uses 16 cores, rest use 1 core.

Fig. 2. *Utility and execution time* of OPT and the heuristics for the *REQ workload*, when varying the drone count m and the activities n.

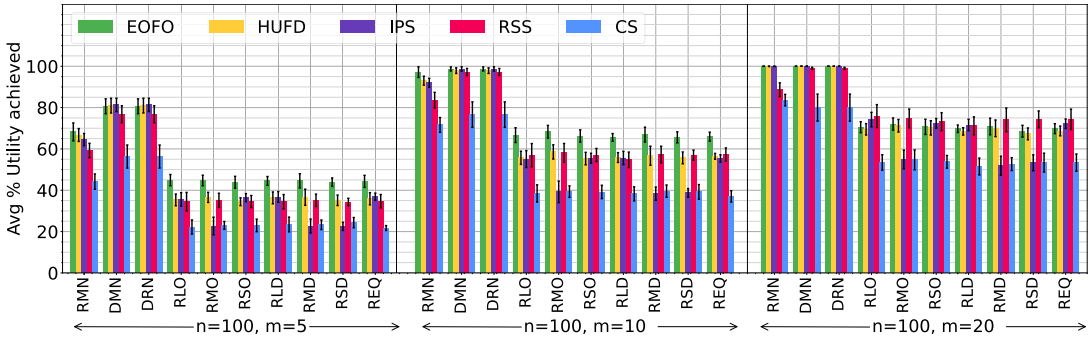


Fig. 3. *Average % of total possible utility achieved* per workload instance by the 5 heuristics for 10 *workload types*, with $n = 100$ activities and $m = \{5, 10, 20\}$ drones for each instance.

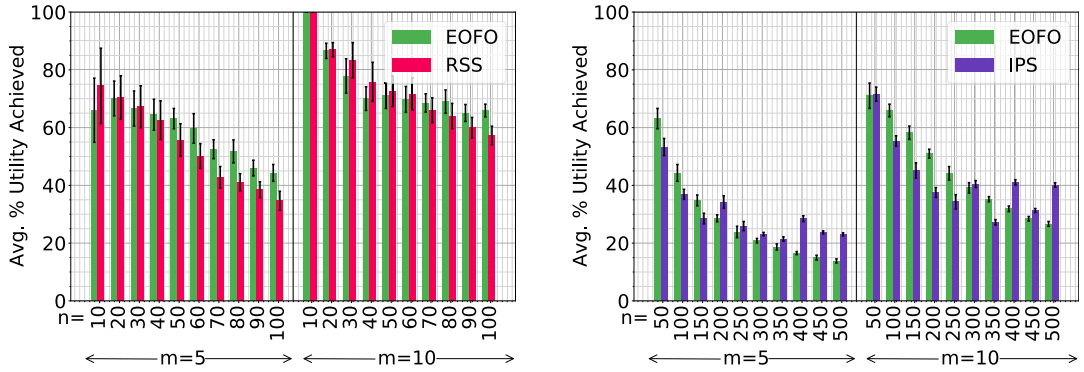
all our proposed workloads, even for the smallest scenario of $m = 5, n = 100$. Hence we limit further analysis only to our heuristic algorithms. Figure 3 plots the *average utility %* achieved by the 5 heuristics for 10 workload types, when using a *fixed* number of activities $n = 100$ and *varying* the number of drones, $m = \{5, 10, 20\}$. The bars are averaged across the 10 workload instances for each type.

The DFS workloads obtain more utility than the RND workloads. Across all heuristics, the utility % achieved by the DFS-based workloads, DMN and DRN, are better than the 7 random-placement workload types with diverse models (RLO to REQ); RMN is slightly worse or comparable to the DFS types (Figure 3). DMN and DRN have activities located in DFS ordering within the road network, which is realistic for many traversal applications. So the activities are spatially and temporally proximate. This makes it more amenable to form single trips with many such waypoint activities present in them. E.g., the average waypoints per trip for DMN and DRN are 5.2 and 5.5, respectively, for $m = 5, n = 100$ while it is only 3.7 for REQ. Among the RND workloads, RMN having just the MNet model gives a higher utility compared to the ones with a mix of model types. Since MNet has a batch execution time of $\rho_M = 11\,\text{s}$, its batches are likely to be processed on-time within the deadline $\delta = 120\,\text{s}$, compared to the longer model execution times of $\rho \in (11, 391)\,\text{s}$ for the diverse workloads, albeit with longer deadlines. E.g., comparing RMN with REQ for $n = 100, m = 5$, across all heuristics, RMN accrues $\approx 14\%$ more *on-time utility* and $\approx 3\%$ higher *on-board utility*.

Most heuristics perform consistently across workloads, with EOFO *often the best.* As we vary the workload types, all heuristics, except IPS, show a similar relative performance trend. In fact, for the 7 workloads, RLO to REQ, all heuristics but IPS exhibit just $\approx$ 0–3% deviation across the different workloads, for a fixed value of (m, n). EOFO usually performs better than the others for most workloads, though the difference may be modest. The EOFO algorithm design uses the *observation start time* to schedule activities, which imposes a natural partial order and yields a sequence of activities to be visited. This sequence improves the data capture utility marginally, e.g., EOFO obtains a 3% better data capture utility compared to RSS for the REQ workload with $n = 100, m = 10$. RSS and HUFD are often competitive with EOFO, and perform better under specific workload conditions, as discussed below.

CS consistently gives the worst utility, being 19% worse on average than the next best heuristic across all workloads. It is unable to identify energy efficient tours and consistently places over 12% fewer activities on tours. However, it is fast, an takes sub-second latency to evaluate.

IPS has a deviation of $\approx 13\%$ in the utility obtained across workloads for a given (m, n). It is at or above the median among all heuristics for all but three workload types: RMO, RMD and RSD, where it is comparable to CS, which gives the lowest utility. In the Interval Partitioning Phase of IPS, the intervals are created based only on energy feasibility and not on the total utility of the interval. This may cause the final utility obtained to vary depending on the workload type. Overall, we see a 7% drop in the data capture utility for the 3 workloads indicating that the intervals did not prioritize utility. However, as discussed later, IPS gives a better utility than EOFO for some scenarios.



(a) EOFO vs. RSS, $m = \{5, 10\}, n = 10 \sim 100$.

(b) EOFO vs. IPS, $m = \{5, 10\}, n = 10 \sim 500$.

Fig. 4. Comparing the performance of heuristics for REQ workload, with varying counts of drones m and activities n.

RSS *outperforms* EOFO *for small load factors.* RSS is the closest competitor to EOFO, and occasionally does better. Figure 4a compares these best-two heuristic algorithms for the REQ workload. The two algorithms exhibit different behaviors as the *load factor* $x = \frac{n}{m}$ varies. For a given drone count, RSS offers a higher % achieved utility than EOFO with fewer number of activities (lower load factor), while EOFO is better for more number of activities (higher load factor). In Figure 3, for $x = \frac{100}{20} = 5$, we see RSS gain $\approx 3\%$ more utility when compared to EOFO. But for a higher load factor $x = \frac{100}{5} = 20$, we see EOFO obtain $\approx 9\%$ more utility than RSS. The two algorithms give similar utilities when $x \approx 6$, e.g., $(m = 5, n = 30)$ and $(m = 10, n = 60)$.

This is also evidenced by Figure 5 which plots the median % utility achieved on the Y-axis for all heuristics against the load factor x on the X-axis, for several (m, n) pairs. For load factors x = 2–6, RSS (red dot) falls on the Pareto boundary along the top-right. We see this behavior persist across different workloads. We theorize that at lower load factors, most activities can be assigned a drone and RSS's scheduling phase yields higher utility than EOFO by including *test and swap assignment* over the *default assignment*. Beyond that threshold, EOFO is not only preferable because it gets more utility, but also because its execution time is smaller. Accordingly, one can choose the best algorithm depending on the drone-activity scenario.

IPS *outperforms* EOFO *under very high load factors* Figure 4b compares the obtained percentage utility by IPS against EOFO for the REQ workload. We see that with $m = 5$ drones and a large number of activities, $n \geq 200$, IPS achieves a higher utility % than EOFO, though it is small at only $\approx 25\%$ of total possible utility being achieved. This is also seen in the higher median utility % seen for IPS in the scatter plot, Figure 5, as the load factor $x \geq 30$.

IPS tries to build a set of conflict-free temporal intervals to assign to each drone. Intuitively, when the number of activities become very large relative to the number of drones, the number of intervals grow. At the same time, the possibility of making a viable *sub-optimal schedule* for each drone also increases dramatically. Hence, IPS is preferable for larger load factors. However, its time complexity depends on the number $\hat{n}$ of intervals, which can be large. Having $n \gg m$ may not be a common scenario, and may lead to incomplete activities.

HUFD *outperforms* EOFO *for workloads with a priority skew.* In the different RND workloads examined so far, HUFD never outperforms EOFO. The HUFD heuristic is designed to prioritize selecting activities with high utilities rather than the observation starting time. Figure 6 plots the obtained % utility for HUFD and EOFO using the PRND workload. In this workload, only 5% of the activities have a high utility, while the rest yield low utility. This characterizes situations in which a few activities have a very high priority – hence high utility, while the rest do offer lower value. We can see that with fewer activities, $n \leq 20$, both the heuristics achieve similar utilities. But with a higher activity count, $n \geq 80$, HUFD obtains $\approx 3\%$ more utility than EOFO. E.g., for $n = 100$, HUFD executes all the high priority activities, while EOFO only executes 80% of them, causing a drop in the utility obtained.

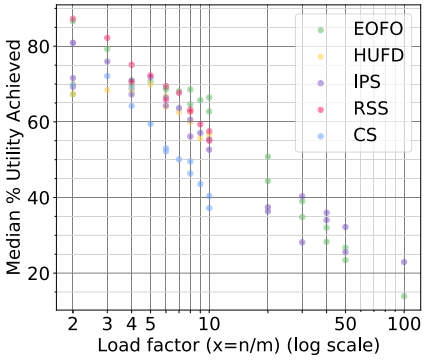


Fig. 5. Median utility % achieved for various load factors ($x = \frac{n}{m}$).

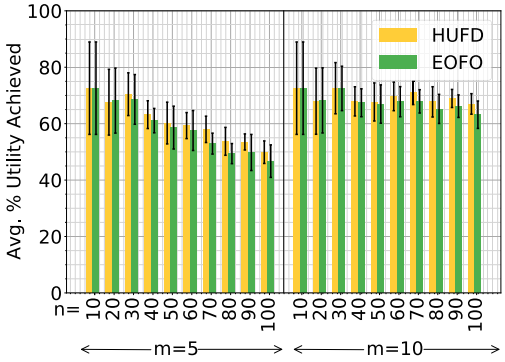


Fig. 6. HUFD vs. EOFO for PRND workload, $m = \{5, 10\}, n = 10 \sim 100$.

3) Some Trips Are Incomplete With Real Drone Traces: The *utilities* reported above are under ideal conditions. Each trip plan generated by a heuristic should complete within a drone's energy capacity. In practice, factors such as non-linear

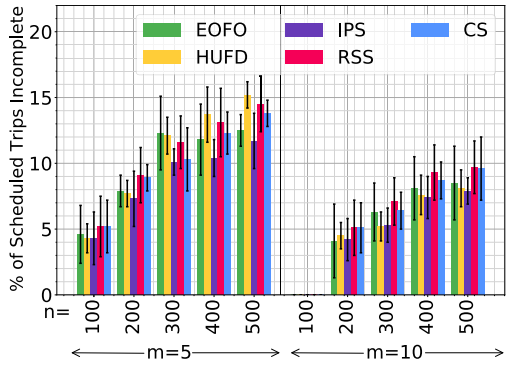


Fig. 7. *% incomplete trips* when using real-world drone energy traces. None of the trips fail for $n \leq 80$.

battery performance can increase or decrease the actual energy consumed. Here, use energy traces from real-world drone flights to estimate the realistic energy usage and the *trip completion rate*.

We run two flight benchmarks on the X-wing quad-copter out-doors: (1) flying at a fixed altitude of $5\,\mathrm{m}$ and speed of $4\,\mathrm{m/sec}$ along the sides of a square of size $10\,\mathrm{m}$, and (2) hovering at a fixed altitude of $5\,\mathrm{m}$. Its camera is turned off and no on-board computation is performed, and this runs until the drone exhausts its battery. We measure the battery energy that is remaining, sampled every $1\,\mathrm{s}$ using on-board sensors. Separately, we also continuously measure the battery usage for the drone when it is turned on but stationary on the ground, while ResNet-18 inference is performed using its on-board computer. These experiments provided us with the traces for the drone's battery consumption when flying, hovering, and performing on-board compute, respectively.

The scheduling heuristics return a timeline for each drone's trip, for a given workload instance. This *trip timeline* contains the time at which the drone should take off, land, as well as the time periods during which it will fly, hover, and perform on-board compute. We replay the drone's trip while using the real-world energy trace to determine the realistic energy usage during these periods of flying, hovering and computing. This helps us check if the battery capacity is exhausted before the trip completes; if so, the trip is *incomplete*, and if not, it can *complete*. This does not consider other ambient factors such as wind-speed, etc.

Figure 7 plots the fraction of trips returned by each scheduling algorithm for a mission, which, according to the drone trace, do not complete. Here we report numbers for $m = 5 \sim 10$ drones and $n = 100 \sim 500$ activities. When the load is for $m \leq 5$ drones and $n \leq 80$ activities, all trips returned by all heuristics can complete within their energy budget using the real-world traces. But with $n > 80$ activities, some trips in the planned schedule start to fail. Even here, only $\approx 15\%$ of trips are incomplete even with $m = 5, n = 500$. So the effect of real-world factors is tangible but minor. By maintaining a buffer battery capacity of $\approx 10 \sim 15\%$ when planning a schedule, we can ensure that the drones can complete a trip and return to the depot. A more robust definition of the MSP problem and system model, to be explored in future, will include a richer energy model that includes the impact of wind, altitude, speed and communications.

VIII. Conclusion

This paper introduces a novel Mission Scheduling Problem (MSP) that co-schedules routes and analytics for drones, maximizing the utility for completing activities. We proposed an optimal algorithm, OPT, and five time-efficient heuristics, CS, RSS, HUFD, EOFO, and IPS. Evaluations using three workloads, varying drone counts and load factors, and real traces exhibit different trade-offs between utility and execution time. OPT is best for small instances, RSS and EOFO are the best in general, and IPS works well when the number of activities is really large. Their time complexity matches reality. The schedules work well for fast and slow DNNs, though on-time utility drops for the latter.

The MSP proposed here is just one variant of an entire class of fleet co-scheduling problems for drones. Other architectures can be explored considering 4G/5G network coverage to send edge results to the back-end, or even off-load captured data to the cloud if it is infeasible to compute on the drone. This will allow more pathways for data sharing among UAVs and GS, but impose energy, bandwidth and latency costs for communications. Even the routing can be aware of cellular coverage to ensure deterministic off-loading on a trip. We can use alternate cost models by assigning an operational cost per trip or per visit, and convert the MSP into a profit maximization problem. The activity time-windows may be relaxed rather than be defined as a static window. Drones with heterogeneous capabilities, in their endurance, compute capabilities, and sensors, will also be relevant for performing diverse activities such as picking up a package using an on-board claw and visually verifying it using a DNN. Finally, we need to deal with dynamics and uncertainties like wind, obstacles and non-linear battery or compute behavior that affect flight paths, energy consumption and utilities. We can use probability distributions and stochastic approaches coupled with real-time information, which can decide and enact on-line rescheduling and rerouting while on a trip. Such on-the-fly route updates for drones allows us to accept and schedule activities continuously, accumulate a mission over hours, and prioritize the profitable activities. These will also need to be validated using more robust real-world experiments and traces.

Acknowledgment

The authors would like to thank the editors and the anonymous reviewers for insightful suggestions that significantly helped improve the technical and presentation quality of the manuscript. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] F. Mohammed, A. Idries, N. Mohamed, J. Al-Jaroodi, and I. Jawhar, "UAVs for smart cities: Opportunities and challenges," in *Proc. Int. Conf. Unmanned Aircr. Syst. (ICUAS)*, May 2014, pp. 267–273.

[2] K. Kanistras, G. Martins, M. J. Rutherford, and K. P. Valavanis, "A survey of unmanned aerial vehicles (UAVs) for traffic monitoring," in *Proc. IEEE Int. Conf. Unmanned Aircr. Syst.*, May 2013, pp. 221–234.

[3] S. George, J. Wang, M. Bala, T. Eiszler, P. Pillai, and M. Satyanarayanan, "Towards drone-sourced live video analytics for the construction industry," in *Proc. 20th Int. Workshop Mobile Comput. Syst. Appl.*, Feb. 2019, pp. 3–8.

[4] F. B. Sorbelli, F. Corò, S. K. Das, and C. M. Pinotti, "Energy-constrained delivery of goods with drones under varying wind conditions," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 9, pp. 6048–6060, Sep. 2021.

[5] F. B. Sorbelli, C. M. Pinotti, S. Silvestri, and S. K. Das, "Measurement errors in range-based localization algorithms for UAVs: Analysis and experimentation," *IEEE Trans. Mobile Comput.*, vol. 21, no. 4, pp. 1291–1304, Apr. 2022.

[6] D. G. Costa and J. P. J. Peixoto, "COVID-19 pandemic: A review of smart cities initiatives to face new outbreaks," *IET Smart Cities*, vol. 2, no. 2, pp. 64–73, Jul. 2020.

[7] M. Gapeyenko, V. Petrov, D. Moltchanov, S. Andreev, N. Himayat, and Y. Koucheryavy, "Flexible and reliable UAV-assisted backhaul operation in 5G mmWave cellular networks," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 11, pp. 2486–2496, Nov. 2018.

[8] S. Jung, S. Hwang, H. Shin, and D. H. Shim, "Perception, guidance, and navigation for indoor autonomous drone racing using deep learning," *IEEE Robot. Autom. Lett.*, vol. 3, no. 3, pp. 2539–2544, Jul. 2018.

[9] Y. Zeng, Q. Wu, and R. Zhang, "Accessing from the sky: A tutorial on UAV communications for 5G and beyond," *Proc. IEEE*, vol. 107, no. 12, pp. 2327–2375, Dec. 2019.

[10] N. H. Motlagh, M. Bagaa, and T. Taleb, "Energy and delay aware task assignment mechanism for UAV-based IoT platform," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 6523–6536, Aug. 2019.

[11] X. Hu, K. Wong, K. Yang, and Z. Zheng, "UAV-assisted relaying and edge computing: Scheduling and trajectory optimization," *IEEE Trans. Wireless Commun.*, vol. 18, no. 10, pp. 4738–4752, Oct. 2019.

[12] A. Trotta, F. D. Andreagiovanni, M. Di Felice, E. Natalizio, and K. R. Chowdhury, "When UAVs ride a bus: Towards energy-efficient city-scale video surveillance," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2018, pp. 1043–1051.

[13] J. Meng, H. Tan, C. Xu, W. Cao, L. Liu, and B. Li, "Dedas: Online task dispatching and scheduling with bandwidth constraint in edge computing," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2019, pp. 2287–2295.

[14] G. Clarke and J. W. Wright, "Scheduling of vehicles from a central depot to a number of delivery points," *Oper. Res.*, vol. 12, no. 4, pp. 568–581, Aug. 1964.

[15] P. Toth and D. Vigo, *The Vehicle Routing Problem*. Philadelphia, PA, USA: SIAM, 2002.

[16] A. S. Manne, "On the job-shop scheduling problem," *Oper. Res.*, vol. 8, no. 2, pp. 219–223, 1960.

[17] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Comput. Surv.*, vol. 31, no. 4, pp. 406–471, Dec. 1999.

[18] A. Khochare, Y. Simmhan, F. B. Sorbelli, and S. K. Das, "Heuristic algorithms for co-scheduling of edge analytics and routes for UAV fleet missions," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, May 2021, pp. 1–10.

[19] M. Jünger, G. Reinelt, and G. Rinaldi, "The traveling salesman problem," in *Handbooks in Operations Research and Management Science*, vol. 7. Amsterdam, The Netherlands: Elsevier, 1995, pp. 225–330.

[20] M. T. Goodrich and R. Tamassia, *Algorithm Design and Applications*, vol. 363. Hoboken, NJ, USA: Wiley, 2015.

[21] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," CMU, Pittsburgh, PA, USA, Tech. Rep. 388, 1976.

[22] O. Cheikhrouhou and I. Khoufi, "A comprehensive survey on the multiple traveling salesman problem: Applications, approaches and taxonomy," *Comput. Sci. Rev.*, vol. 40, May 2021, Art. no. 100369.

[23] J. K. Lenstra and A. H. G. R. Kan, "Complexity of vehicle routing and scheduling problems," *Networks*, vol. 11, no. 2, pp. 221–227, 1981.

[24] G. Desaulniers, F. Errico, S. Irnich, and M. Schneider, "Exact algorithms for electric vehicle-routing problems with time windows," *Oper. Res.*, vol. 64, no. 6, pp. 1388–1405, Dec. 2016.

[25] E. Uchoa, D. Pecin, A. Pessoa, M. Poggi, T. Vidal, and A. Subramanian, "New benchmark instances for the capacitated vehicle routing problem," *Eur. J. Oper. Res.*, vol. 257, no. 3, pp. 845–858, Mar. 2017.

[26] D. Cattaruzza, N. Absi, and D. Feillet, "Vehicle routing problems with multiple trips," *Ann. Oper. Res.*, vol. 271, no. 1, pp. 127–159, Dec. 2018.

[27] F. Stavropoulou, P. P. Repoussis, and C. D. Tarantilis, "The vehicle routing problem with profits and consistency constraints," *Eur. J. Oper. Res.*, vol. 274, no. 1, pp. 340–356, Apr. 2019.

[28] S. P. Gayialis, G. D. Konstantakopoulos, G. A. Papadopoulos, E. Kechagias, and S. T. Ponis, "Developing an advanced cloud-based vehicle routing and scheduling system for urban freight transportation," in *Proc. IFIP Int. Conf. Adv. Prod. Manage. Syst.* Cham, Switzerland: Springer, 2018, pp. 190–197.

[29] K. Fagerholt, "Optimal fleet design in a ship routing problem," *Int. Trans. Oper. Res.*, vol. 6, no. 5, pp. 453–464, Sep. 1999.

[30] I. Khoufi, A. Laouiti, and C. Adjih, "A survey of recent extended variants of the traveling salesman and vehicle routing problems for unmanned aerial vehicles," *Drones*, vol. 3, no. 3, p. 66, Aug. 2019.

[31] E. Balas, G. Lancia, P. Serafini, and A. Vazacopoulos, "Job shop scheduling with deadlines," *J. Combinat. Optim.*, vol. 1, no. 4, pp. 329–353, 1998.

[32] H. Mokhtari and M. Dadgar, "Scheduling optimization of a stochastic flexible job-shop system with time-varying machine failure rate," *Comput. Oper. Res.*, vol. 61, pp. 31–45, Sep. 2015.

[33] C. Le Pape and P. Baptiste, "Resource constraints for preemptive job-shop scheduling," *Constraints*, vol. 3, no. 4, pp. 263–287, 1998.

[34] P. Varshney and Y. Simmhan, "Characterizing application scheduling on edge, fog, and cloud computing resources," *Softw., Pract. Exper.*, vol. 50, no. 5, pp. 558–595, May 2020.

[35] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, 3rd Quart., 2017.

[36] J. Feng, Z. Liu, C. Wu, and Y. Ji, "Mobile edge computing for the Internet of Vehicles: Offloading framework and job scheduling," *IEEE Veh. Technol. Mag.*, vol. 14, no. 1, pp. 28–36, Mar. 2019.

[37] Z. Ning, J. Huang, X. Wang, J. J. P. C. Rodrigues, and L. Guo, "Mobile edge computing-enabled Internet of Vehicles: Toward energy-efficient scheduling," *IEEE Netw.*, vol. 33, no. 5, pp. 198–205, Sep. 2019.

[38] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura, "Serendipity: Enabling remote computing among intermittently connected mobile devices," in *Proc. 13th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2012, pp. 145–154.

[39] J. Zhang et al., "Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 2633–2645, Aug. 2018.

[40] J. Feng, Z. Liu, C. Wu, and Y. Ji, "AVE: Autonomous vehicular edge computing framework with ACO-based scheduling," *IEEE Trans. Veh. Technol.*, vol. 66, no. 12, pp. 10660–10675, Dec. 2017.

[41] J. Yang, X. You, G. Wu, M. M. Hassan, A. Almogren, and J. Guna, "Application of reinforcement learning in UAV cluster task scheduling," *Future Gener. Comput. Syst.*, vol. 95, pp. 140–148, Jun. 2019.

[42] C. Wang, D. Deng, L. Xu, and W. Wang, "Resource scheduling based on deep reinforcement learning in UAV assisted emergency communication networks," *IEEE Trans. Commun.*, vol. 70, no. 6, pp. 3834–3848, Jun. 2022.

[43] A. Ferdowsi, M. A. Abd-Elmagid, W. Saad, and H. S. Dhillon, "Neural combinatorial deep reinforcement learning for age-optimal joint trajectory and scheduling design in UAV-assisted networks," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 5, pp. 1250–1265, May 2021.

[44] X. Chen et al., "DeliverSense: Efficient delivery drone scheduling for crowdsensing with deep reinforcement learning," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput.*, Sep. 2022, pp. 11–15.

[45] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, "Quantifying generalization in reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 1282–1289.

[46] R. G. Baraniuk, V. Cevher, M. F. Duarte, and C. Hegde, "Model-based compressive sensing," *IEEE Trans. Inf. Theory*, vol. 56, no. 4, pp. 1982–2001, Apr. 2010.

[47] H. V. Abeywickrama, B. A. Jayawickrama, Y. He, and E. Dutkiewicz, "Empirical power consumption model for UAVs," in *Proc. IEEE 88th Veh. Technol. Conf. (VTC-Fall)*, Aug. 2018, pp. 1–5.

[48] Y. Hu, J. C. S. Lui, W. Hu, X. Ma, J. Li, and X. Liang, "Taming energy cost of disk encryption software on data-intensive mobile devices," *Future Gener. Comput. Syst.*, vol. 107, pp. 681–691, Jun. 2020.

[49] S. Bianco, R. Cadene, L. Celona, and P. Napoletano, "Benchmark analysis of representative deep neural network architectures," *IEEE Access*, vol. 6, pp. 64270–64277, 2018.

[50] R. D. Yates, Y. Sun, D. R. Brown, S. K. Kaul, E. Modiano, and S. Ulukus, "Age of information: An introduction and survey," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 5, pp. 1183–1210, May 2021.

[51] A. Olivera and O. Viera, "Adaptive memory programming for the vehicle routing problem with multiple trips," *Comput. Operations Res.*, vol. 34, no. 1, pp. 28–47, Jan. 2007.

[52] R. L. Graham, "Bounds for certain multiprocessing anomalies," *Bell Syst. Tech. J.*, vol. 45, no. 9, pp. 1563–1581, Nov. 1966.

[53] B. Zmazek, "Multiple trips within working day in capacitated VRP with time windows," in *Proc. 10th WSEAS Int. Conf. Comput.*, 2006, pp. 93–99.

[54] D. Birant and A. Kut, "ST-DBSCAN: An algorithm for clustering spatial–temporal data," *Data Knowl. Eng.*, vol. 60, no. 1, pp. 208–221, Jan. 2007.

[55] G. Dósa and J. Sgall, "First fit bin packing: A tight analysis," in *Proc. Int. Symp. Theor. Aspects Comput. Sci.*, 2013, p. 538.

[56] J.-Y. Potvin and J.-M. Rousseau, "An exchange heuristic for routeing problems with time windows," *J. Oper. Res. Soc.*, vol. 46, no. 12, pp. 1433–1446, Dec. 1995.
[57] E. Fix and J. L. Hodges Jr., "Discriminatory analysis. Nonparametric discrimination: Consistency properties," *Int. Stat. Rev.*, vol. 57, no. 3, pp. 238–247, 1989.
[58] F. Gavril, "Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph," *SIAM J. Comput.*, vol. 1, no. 2, pp. 180–187, Jun. 1972.
[59] Y. D. Liang, S. K. Dhall, and S. Lakshmivarahan, "On the problem of finding all maximum weight independent sets in interval and circular-arc graphs," in *Proc. Symp. Appl. Comput.*, Jan. 1991, pp. 465–466.
[60] T. Erlebach and F. C. Spieksma, "Simple algorithms for a weighted interval selection problem," in *Proc. Int. Symp. Algorithms Comput.* Cham, Switzerland: Springer, 2000, pp. 228–240.
[61] IBM ILOG, "V12.1: User's manual for CPLEX," Int. Bus. Mach. Corp., Armonk, NY, USA, Tech. Rep. 53, 2009.
[62] O. Wiki. (2020). *India—Openstreetmap Wiki*. [Online]. Available: https://wiki.openstreetmap.org/w/index.php?title=India
[63] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.
[64] J. Wang et al., "Bandwidth-efficient live video analytics for drones via edge computing," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Oct. 2018, pp. 159–173.
[65] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 3431–3440.
[66] H. Perreault, G. Bilodeau, N. Saunier, and M. Héritier, "SpotNet: Self-attention multi-task network for object detection," in *Proc. 17th Conf. Comput. Robot Vis. (CRV)*, May 2020, pp. 230–237.
[67] L. Wen et al., "Drone-based joint density map estimation, localization and tracking with space-time multi-scale attention network," 2019, *arXiv:1912.01811*.
[68] B. K. P. Horn and B. G. Schunck, "Determining optical flow," *Artif. Intell.*, vol. 17, nos. 1–3, pp. 185–203, Aug. 1981.

Aakash Khochare is currently pursuing the Ph.D. degree with the Indian Institute of Science, Bengaluru, India. His research interests include designing systems, abstractions, heuristics that enable analytics on edge, fog, and drone platforms. He was a recipient of the IEEE TCSC SCALE Challenge Award in 2019.

Francesco Betti Sorbelli (Member, IEEE) received the bachelor's and master's degrees (Hons.) in computer science from the University of Perugia in 2007 and 2010, respectively, and the Ph.D. degree in computer science from the University of Florence in 2018. He was a Post-Doctoral Researcher with the Missouri University of Science and Technology in 2020 and the University of Perugia in 2018, 2021, and 2022. He is currently an Assistant Professor with the University of Perugia, Italy. His research focuses on algorithm design, combinatorial optimization, and unmanned vehicles.

Yogesh Simmhan (Senior Member, IEEE) received the Ph.D. degree in computer science from Indiana University, Bloomington. He is currently an Associate Professor and a Swarna Jayanti Fellow with the Indian Institute of Science, Bengaluru. His research explores abstractions, algorithms, and applications on distributed systems to support big data and IoT applications. He is also the Associate Editor-in-Chief of the *Journal of Parallel and Distributed Systems* and an Associate Editor of *Future Generation Computer Systems*. Previously, he was a Research Faculty Member with the University of Southern California (USC) and a Post-Doctoral Researcher with Microsoft Research. He is a Distinguished Contributor of IEEE CS and a Distinguished Member of ACM.

Sajal K. Das (Fellow, IEEE) is currently a Curator's Distinguished Professor in computer science and the Daniel St. Clair Endowed Chair with the Missouri University of Science and Technology, Rolla, USA. His research interests include wireless sensor networks, mobile and pervasive computing, cyber-physical systems and IoT, cloud computing, and cyber security. He serves as the Founding Editor-in-Chief for *Pervasive and Mobile Computing* (Elsevier) and as an Associate Editor of several journals, including the IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE/ACM TRANSACTIONS ON NETWORKING, and *ACM Transactions on Sensor Networks*.