Heuristic Algorithms for Co-scheduling of Edge Analytics and Routes for UAV Fleet Missions

Aakash Khochare*, Yogesh Simmhan*, Francesco Betti Sorbelli†, and Sajal K. Das†
*Department of Computer Science, Indian Institute of Science, India. Email: {aakhochare, simmhan}@iisc.ac.in
†Dept. of Computer Science, Missouri Univ. of Science & Tech., USA. Email: {francesco.bettisorbelli, sdas}@mst.edu

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 utility from the activities while meeting activity deadlines as well as 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 two efficient heuristic algorithms, JSC and VRC, that provide fast sub-optimal solutions. Evaluation of these three 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, path planning.

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 (including COVID-19) 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. Usually, such activities involve hovering and recording a scene using the drone’s camera, and analyzing the videos to take decisions.

Advancements of computer vision algorithms and Deep Neural Networks (DNNs) enable video analytics to be performed over such recordings for automated decision-making. Typically, these are inferred once the recordings 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. Hence, the on-board edge computing capability [8] available on commercial drones can be leveraged to process the recorded videos, and quickly report concise results to the GS over 3/4/5G wireless networks [9]. Since the transferred results are brief and the on-board processing times dominate, we ignore communication constraints 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 exhausted 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], and 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]. However, 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.

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].

We make the following contributions in this paper.

- 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 (Sections III and IV).

- We prove that MSP is NP-hard, and optimally solve it using a mixed integer linear programming (MILP) design, OPT, which is feasible for small inputs (Section V).
- We design two time-efficient heuristic algorithms, JSC and VRC, that solve the MSP for arbitrary-sized inputs, and offer complexity bounds for their execution (Section VI).
- We evaluate and analyze the utility and scheduling runtime trade-offs for these three algorithms, for diverse drone workloads based on real drone traces (Section VII).

II. RELATED WORK

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

A. Vehicle Routing Problem (VRP)

VRP is a variant of TSP with multiple salespersons [14] and it is NP-hard [18]. This problem has had several extensions to handle realistic scenarios, such as temporal constraints that impose deliveries only at specific time-windows [19], capacity constraints on vehicle payloads [20], multiple trips for vehicles [21], profit per vehicle [22] and traffic congestion [23]. VRP has also been adapted for route planning for a fleet of ships [24], and for drone assisted delivery of goods [25].

In [10] the scheduling of events is performed by UAVs at specific locations, involving data sensing/processing and communication with the GS. The goal here is to minimize the drone’s energy consumption and operation time. Factors like wind and temperature that may affect the route and execution time are also considered. While they combine sensing and processing into one monolithic event, these are 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.

In [11] the use of UAVs is explored 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.

Trotta et al. [12] propose a novel architecture for energy-efficient video surveillance of points of interest (POIs) in a city by drones. The UAVs use bus rooftops for re-charging and being transported to the next POI based on known bus routes. Drones also act as relays for other drones capturing videos. The mapping of drones to bus routes is formulated as an MILP problem and a TSP-based heuristic is proposed. Unlike ours, their goal is not to schedule and process data onboard the drone. Similarly, we do not examine any data off-loading from the drone, nor any piggy-backing mechanisms.

B. Job-shop Scheduling (JSC)

Scheduling of computing tasks on drones is closely aligned with scheduling tasks on edge and fog devices [26], and broadly with parallel workload scheduling [17] and JSC [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 et al. [27] propose a framework for cooperative edge computing on autonomous road vehicles, which aims to increase their decentralized computational capabilities and task execution performance. Others [28] combine optimal placement of data blocks with optimal task scheduling to reduce computation delay and response time for the submitted tasks while improving user experience in edge computing. In contrast, we co-schedule UAV routing and edge computing.

There exist works that explore task scheduling for mobile clients, and off-load computing to another nearby edge or fog resource. These may be categorized based on their use of predictable or unpredictable mobility models. In [29], the mobility of a vehicle is predicted and used to select the roadside edge computing unit to which the computation is off-loaded. Serendipity [30] takes an alternate view and assumes that mobile edge devices interact with each other intermittently and at random. This makes it challenging to determine if tasks should be off-loaded to another proximate device for reliable completion. The problem we solve 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. [31], 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.

III. MODELS AND ASSUMPTIONS

This section introduces the UAV system model, application model, and utility model along with underlying assumptions.

A. UAV System Model

Let $\lambda = (0,0,0)$ 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 drones. For simplicity, we assume that all the drones are homogeneous. Each drone has a camera for recording videos, which is subsequently processed. This processing can be done using the on-board computing, or done offline once the drone lands (which is outside the scope of our problem). The on-board processing speed is $\pi$ 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 $\epsilon^f$ be the energy required for flying for a unit time duration at a constant energy-efficient speed $s$ within the Cartesian space; let $\epsilon^h$ be the energy for hovering for a unit time duration; and let $\epsilon^c$ be the energy for performing computation for a unit time duration. For simplicity, we ignore the energy for video capture since it is negligible in practice. Also, a drone that returns to the depot for performing computation for a unit time duration. For hovering for a unit time duration; and let $\epsilon$ be the energy for an activity $\alpha$ is common for analytics over constrained resources [32].

$\lambda$ given the processing speed all batches of an activity. So, the processing time may be executed at a time on-board a drone and it should for simplicity, we discretize all time-units into integers.

Fig. 1. Sample MSP scenario. a) shows a city with the depot ($\lambda$); 6 waypoints to visit ($\lambda_i$) with some utility; and possible trip routes for drones ($R^1_i$). b) has the corresponding 6 activities ($\alpha_i$) with data capture duration (shaded) and compute deadline (vertical line) and the two available drones.

B. Application Model

Let $A = (\alpha_1, \ldots, \alpha_n)$ be the set of $n$ activities to be performed starting from time $t = 0$, where each activity $\alpha_i$ is given by the tuple $(\lambda_i, t_i, t_i, \gamma_i, \gamma_i, \gamma_i)$. 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 $\lambda$. The starting and ending times for performing the data capture task are $t_i$ and $t_i$. The compute requirements for subsequently processing all of the captured data is $\kappa_i$ floating point operations. Lastly, $\delta_i$ is the time deadline by which the computation task should be completed on the drone to derive on-time utility of processing, while $\gamma_i$, $\gamma_i$, and $\gamma_i$ are the data capture, on-time processing and on-board processing utility values that are gained for completing the activity. These are described in the next sub-section.

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 [32]. Specifically, for an activity $\alpha_i$, the data captured between $(t_i - t_i)$ is divided into batches of a fixed duration $\beta$, with the sequence of batches given by $B_i = (b_i, \ldots, b_i)$, where $q_i = |B_i| = \left\lceil \frac{t_i - t_i}{\beta} \right\rceil$. The computational cost to process each batch is $\kappa_i b_i = \frac{\kappa_i}{\rho_i}$ floating-point operations, and is constant for all batches of a previous batch of the same or a different activity. All batches for a single activity should be executed in sequence, i.e., complete processing $b_i$ before processing $b_i+1$. Once a batch is processed, its compact results are immediately and deterministically communicated to the GS.

C. Utility Model

The primary goal of the drone is to capture videos at the various activity locations for the specified duration. This is a necessary condition for an activity to be successful. We define this as the data capture utility $\gamma_i$ accrued by a drone for an activity $\alpha_i$. 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 $\gamma_i$. Other activities may be time-sensitive and have a soft-deadline $\delta_i$ for completing the processing. For these, if we process its captured data on the drone by this deadline, we receive an extra on-time processing utility $\gamma_i$. The processing utilities accrue pro rata, for each batch of the activity completed.

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 to the drones, within the energy and compute constraints of the drones, such that the total utility achieved is maximized. It is 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 using the whole fleet to maximize the utility. There is a fixed cost 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 $d_j$ 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 $A_j^r = (\alpha_{j1}, \ldots, \alpha_{jn}) \subseteq A$ where $\alpha_{js} \in A$, $j_n \leq n$, and no activity appears twice within a mission. Further, we have $\alpha_{js} < \alpha_{js+1}$, i.e., the observation start and end times of an activity in the mission sequence fully precede those of the next activity in it, $t^r_{js} \leq t^r_{js+1}$. Also, $A_j^r \cap A_k^r = \emptyset \forall j, k, x, y$ to ensure that an activity is mapped 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 $R_j^r = (\lambda_j^r, \lambda_{j1}^r, \ldots, \lambda_{jn}^r, \lambda_j)$, where the starting and ending waypoints of the drone are the depot location $\lambda_j$, and each intermediate location corresponds to the video capture location $\lambda_{jr}$ for the activity $\alpha_{jr}$ in the mission sequence. For uniformity, we denote the first and the last depot location in the route as $\lambda_{j0}^r$ and $\lambda_{jn+1}^r$, respectively. Clearly, $|R_j^r| = j_n + 2$.

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, $\lambda_{jr}$, at departure time $t_j^r$ and reach the next waypoint location, $\lambda_{jr+1}$, at arrival time $\tau_j^r$. Let the function $F(\lambda_{r, \lambda_{r+1}})$ give the flying time between $\lambda_r$ and $\lambda_{r+1}$. Since the drone has a constant flying speed, we have $\tilde{\tau}_{jr+1} = \tau_j^r + F(\lambda_{jr}, \lambda_{jr+1})$.

The drone must hover at each waypoint $\lambda_{jr}$ between $t_j^r$ and $\tilde{\tau}_{jr}$ while recording the video, and it departs the waypoint after this, i.e., $\tau_j^r = \tilde{\tau}_{jr}$. If the drone arrives at this waypoint at time $\tilde{\tau}_{jr}$, i.e., before the observation start time $t_j$, it hovers here for a duration of $\tilde{\tau}_{jr} - t_r$, and then continues hovering during the activity’s video capture. If a drone arrives at $\lambda_{jr}$ after $\tilde{\tau}_{jr}$, it is invalid since the video capture for the activity cannot be conducted for the whole duration. So, $\tilde{\tau}_{jr} \leq t_j^r \leq \tau_j^r$. Also, since the deadline for on-time computation over the captured data is $\delta_j$, we require $\delta_j \geq \tilde{\tau}_{jr}$. Once the drone finishes capturing video for the last activity in its $r^{th}$ trip, it returns back to the depot location at time $\tilde{\tau}_{jr+1} = \tau_j^r + F(\lambda_{jr}, \lambda_j)$. Hence, the total flying time for a drone $d_j$ for its $r^{th}$ trip is:

$$f_j^r = \sum_{i=0}^{n} (\tilde{\tau}_{i,j+1} - \tau_{i,j})$$

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

$$h_j^r = \sum_{i=1}^{n} (t_{i,j}^r - \tilde{\tau}_{i,j}) + \sum_{i=1}^{n} (\tilde{\tau}_{i,j} - t_{i,j}^r) = \sum_{i=1}^{n} (t_{i,j}^r - \tau_{i,j})$$

which includes hovering due to early arrival at a waypoint, and hovering during the data capture.

Let the scheduler assign the time slot $[\theta_j^k, \tilde{\theta}_j^k)$ for executing a batch $b_j^k$ of activity $\alpha_j$, on drone $d_j$, where $\tilde{\theta}_j^k = \theta_j^k + h_j^k$.

Based on the batch execution time, we define a completion function for each activity $\alpha_j$, for the three utility values:

- The data capture completion $\mu_j^k$ for $1$ if the drone hovers at location $\lambda_j$ for the entire period from $t_{i,j}$ to $\tilde{\tau}_{i,j}$, and is $0$ otherwise.
- The on-board completion $0.0 \leq \bar{\mu}_j^k \leq 1.0$ indicates the fraction of batches of that activity that are completed on-board the drone. Let $\mu_j^k = 1$ if the batch $b_j^k$ of activity $\alpha_j$ is completed on-board, and $\bar{\mu}_j^k = 0$ if it is not completed on-board the drone. Then, $\bar{\mu}_j^k = \sum \bar{\mu}_j^k$.
- The on-time completion $0.0 \leq \bar{\mu}_j^k \leq 1.0$ gives the fraction of batches of that activity that are fully completed within the deadline. As before, let $\mu_j^k = 1$ if the batch $b_j^k$ of activity $\alpha_j$ is completed on-time, i.e., $\delta_j \leq \tilde{\tau}_{i,j}$, and $\bar{\mu}_j^k = 0$ otherwise. So, $\bar{\mu}_j^k = \sum \bar{\mu}_j^k$.

The total utility for an activity $\alpha_j$ is $U_j = u_j \gamma_j + u_j \gamma_j + u_j \gamma_j$, and the total computation time of batches on a drone $d_j$ is:

$$c_j = \sum_{\alpha_i \in A} (\mu_j^k + \bar{\mu}_j^k) \cdot \rho_j^k$$

B. Optimization of MSP

Based on these, the objective of the optimization is

$$\text{arg max} \sum_{\alpha_i \in A} U_j, \text{i.e., assign drones to activity waypoints and activity batches to the drones’ computing slots to maximize the utility from data capture, on-board and on-time computation. }$$

These are subject to the following constraints on the execution slot assignments for a batch on a drone:

$$(t_{i,j} + k \cdot \beta) \leq \theta_j^k \leq \theta_j^k + 1, \quad \bar{\theta}_j^k \leq \tilde{\tau}_{i,j+1} \leq \tilde{\tau}_{i,j+1} \leq \bar{\theta}_j^k$$

i.e., the data capture for a duration of $\beta$ for the $k^{th}$ batch of the activity is completed before the execution slot of the batch starts; the batches for an activity are executed in sequence; and the execution completes before the drone lands.

Also, there can only be one batch executing at a time on a drone. So $\forall [\theta_j^k, \tilde{\theta}_j^k)$ and $[\theta_j^k, \tilde{\theta}_j^k)$ slots assigned to batches $b_j^k$ and $b_j^k$ on drone $d_j$, we have $[\theta_j^k, \tilde{\theta}_j^k) \cap [\theta_j^k, \tilde{\theta}_j^k) = \emptyset$.

Lastly, the energy expended by drone $d_j$ on the $r^{th}$ trip, to fly, hover, and compute, should be within its battery capacity:

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

Model Applicability: Our novel model can be abstracted to describe diverse applications. In entity localization [33], $\gamma_i = 0$ and $\tilde{\gamma}_i > 0$ captures the importance of an entity being tracked.

In traffic monitoring [2] it is useful to have timely insights, appropriately tuning $\gamma_i$ and $\tilde{\gamma}_i$. In construction survey [3] there are no strict time deadlines, so $\gamma_i = 0$.

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 SCHEDULING (OPT) based on MILP.
TABLE I
CONSTRAINTS FOR OPT MILP FORMULATION.

| C. | Expression | Meaning |
|----|------------|---------|
| 1  | \( \sum_{k \in D} \sum_{i \in V} \sum_{j \in V} x_{ij}^k \leq 1 \), \( \forall i \in V' \) | The waypoint for an activity \( \alpha_i \) is visited only once. |
| 2  | \( \sum_{k \in D} x_{ij}^k - \sum_{k \in D} x_{ji}^k = 0 \), \( \forall k \in D, l \in \mathbb{R} \) | A drone trip \( l \) starting from the depot must also end there. |
| 3  | \( \sum_{k \in D} x_{ij}^k = 1 \iff \sum_{k \in D} x_{ji}^k = 1 \), \( \forall i \in V', k \in D, l \in \mathbb{R} \) | A drone \( k \) must visit at least one waypoint on each trip \( l \). |
| 4  | \( \sum_{k \in D} x_{ij}^k - \sum_{k \in D} x_{ji}^k = 0 \), \( \forall k \in D, j \in V', l \in \mathbb{R} \) | A drone \( k \) visiting waypoint \( j \) must also fly out from there. |
| 5  | \( (t_j - F_{ij}) \cdot \sum_{k \in D} \sum_{i \in V} x_{ij}^k \geq 0 \), \( \forall j \in V' \) | Any drone flying to waypoint \( j \) from the depot must reach its observation start time \( t_j \). |
| 6  | \( (t_j - t_i - F_{ij}) \cdot \sum_{k \in D} \sum_{i \in V} x_{ij}^k \geq 0 \), \( \forall i \in V', j \in V' \) | Any drone flying to waypoint \( j \) from \( i \) must reach before its observation start time \( t_j \). |
| 7  | \( x_{ik}^{l+1} = \sum_{k \in V} x_{ik}^l \cdot (t_i + F_{ki}) \), \( \forall k \in D, l \in \mathbb{R} \) | Decides the landing time of drone \( k \) at the depot after trip \( l \). |
| 8  | \( x_{ik}^{l+1} \leq \tau_{max} \), \( \forall k \in D, l \in \mathbb{R} \) | Depot landing times for all trips is within the maximum time. |

A. NP-hardness of MSP

As discussed earlier, the MSP combines elements of the VRP and the JSP in assigning routes and batches to drones, for maximizing the overall utility, subject to energy constraints.

Theorem 1. MSP is NP-hard.

Proof. The VRP is NP-hard [18]. In addition, MSP considers multiple-trips, time-windows, energy-constraints, and utilities.

The VRP variant 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 the number of waypoints, \( m = n \), and setting the time horizon \( T_h = \sum_{e \in \mathcal{E}} T(e) \), where \( \mathcal{E} \) is the set of edges and \( T(e) \) is the flying time for traversing an edge [34], and limiting the number of trips to one. The VRP variant 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]. Clearly, a VRP variant with energy-constrained vehicles is still NP-hard, by just relaxing those constraints to match VRP.

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 [21], 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 [35]. 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 SCHEDULING (OPT) algorithm offers an optimal solution to MSP by modeling it as a multi-commodity flow problem (MCF), similar to [39, 12]. We reformulate the MSP definition as an MILP formulation.

The paths in the city are modeled as a complete graph, \( \mathcal{G} = (V, \mathcal{E}) \), between the \( n \) activity waypoint vertices, \( V = \{0, 1, \ldots, n\} \), where 0 is the depot \( \lambda \). Let \( i, j \) and \( i \rightarrow j \) be the set of out-edges and in-edges of a vertex \( i \), and \( V' = V \setminus \{0\} \) be the set of all waypoint vertices. We enumerate the \( m \) drones as \( D = \{1, \ldots, m\} \). Let \( r_{max} \) be the maximum time for completing all the missions, and \( r_{max} \) the maximum trips a drone can do. Let \( \mathcal{R} = \{1, \ldots, r_{max}\} \) be the possible trips.

Let \( x_{ij}^k \in \{0, 1\} \) be a decision variable that equals 1 if the drone \( k \in D \) in its trip \( l \in \mathcal{R} \) traverses the edge \((i, j)\), and 0 otherwise. If \( x_{ij}^k = 1 \), then the waypoint for activity \( \alpha_i \) was visited by drone \( k \) on trip \( l \). Let \( B_i = \{0, \ldots, q_i\} \) be the set of batches of activity \( \alpha_i \). Let \( w_{\alpha_i}^q \) be a binary decision variable used to linearize the batch computation whose value is 1 if batch \( b_{\alpha_i}^q \) is processed before \( \tau_{\alpha_i}^l \), and 0 otherwise [16].

Let \( w_{\alpha_i}^q \) be a decision variable that equals 1 if the drone \( k \in D \) in trip \( l \in \mathcal{R} \) processes the batch \( g \) of activity \( \alpha_i \), within its deadline \( \delta_i \), and 0 otherwise; and similarly, \( z_{ij}^l \) equals 1 if the batch is processed before the drone completes the trip and lands, and 0 otherwise. Let the per batch utility for on-board completion be \( \Gamma_i = \frac{\bar{t}_i}{q_i} \), and on-time completion be \( \Gamma_i = \frac{\bar{t}_i}{q_i} \) for activity \( \alpha_i \). Finally, let \( M \) be a sufficiently large constant.

Using these, the MILP objective is:
\[
\max \sum_{k \in D} \sum_{i \in R} \sum_{u \in U} \left( \sum_{j \in T} x_{ij}^k \cdot \Gamma_i \right) + \left( \sum_{g \in B_i} y_{ig}^k \cdot \Gamma_i + z_{ig}^k \cdot \gamma_i \right)
\]

subject to the constraints listed in Table I.

VI. Heuristic Algorithms for MSP

Since MSP is NP-hard, opt is tractable only for small-sized inputs. So, time-efficient but sub-optimal algorithms are necessary for larger-sized inputs. In this section, we propose two heuristic algorithms, called JOB SCHEDULING CENTRIC (JSC) and VEHICLE ROUTING CENTRIC (VRC).

A. The JSC Algorithm

The JOB SCHEDULING CENTRIC (JSC) algorithm aims to find near-optimal scheduling of batches while ignoring the optimizations of routing to conserve energy. JSC is split into two phases: clustering and scheduling.

1) Clustering Phase: First, we use the ST-DBSCAN algorithm [37] to find time-efficient spatio-temporal clusters of activities. It returns a set of clusters \( C \) such that for activities within a cluster \( C_i \in C \), certain spatial and temporal distance thresholds are met. Drones are then allocated to clusters depending on their availability. For each cluster \( C_i \), let \( T_U^i = \max_{a_j \in C_i} (t_j + F(\lambda_j, \lambda)) \) be the upper bound for the latest landing time for a drone servicing activities in \( C_i \); analogously, let \( T_L^i = \min_{a_j \in C_i} (t_j - F(\lambda_j, \lambda_j)) \) be the lower bound for the earliest take-off time. Then, all the temporal windows \( [T_L^i, T_U^i] \) for each \( C_i \in C \) are sorted with respect to \( T_L^i \). Recalling that there are \( m \) drones available at \( t = 0 \), they 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 \( C_1 \) at time \( T_L^1 \) and released at time \( T_U^1 \); \( c_2 = \frac{m-c_1}{|C_2|} \) allocated to \( C_2 \) from \( T_L^2 \) to \( T_U^2 \) (assuming \( T_L^2 < T_U^3 \)), and so on.

2) Scheduling Phase: Here, the activities are assigned to drones. The feasibility of assigning \( a_i \) to \( d_j \), is tested by checking if the required flying and hovering energy is enough to visit \( A_j \cup A_i \); 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 \( B_i \subseteq B \) within the energy requirements. Assignments are done in two steps: default assignment and test and swap assignment.

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

Test and Swap Assignment. If the 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 \( P^+ \) be the union of the preferred intervals forming the total preferred interval for an activity \( a_i \). Each batch \( b^k \) is tested for violating its deadline. If it violates, then batches \( b^k \) from other activities already scheduled in \( P^+ \) are identified and tested if they too violate their deadline. If so, \( b^k \) is moved to the next available slot in \( S_{b^k} \), and its old time slot given to \( b^k \). If \( b^k \) is in its preferred interval but has more slots available in this interval, then \( b^k \) is moved to another free slot in \( P_{b^k} \) and \( b^k \) assigned to the slot that is freed. Else, the current configuration does not contain violations, except for the current batch \( b^k \), but all available slots are occupied. So, the utility for \( b^k \) is compared with another \( b^k \) in \( P^+ \), and the batch with a higher utility gets this slot.

3) The Core of JSC: The JSC algorithm works as follows (Algorithm 1). After the initial clustering phase, activities are tested for their feasibility. If so, the default assignment is initially evaluated in terms of total utility. If this creates a deadline violation, the test and swap assignment performed, and the best scheduling is applied.

Algorithm 1: JSC(A, D)
1. \( C \leftarrow \) clustering phase
2. for \( C_k \in C \) do
3. for \( \alpha_i \in C_k \) do
4. for \( d_j \) assigned to \( C_k \) do
5. if \( \alpha_i \cup A_i \) is feasible then
6. apply best scheduling among default and test and swap assignment on \( B_i \)

4) Time Complexity of JSC: ST-DBSCAN’s time complexity is \( O(n \log n) \) for \( n \) waypoints. Unlike \( k \)-means clustering, ST-DBSCAN automatically picks a suitable number of clusters, \( k \), with \( \approx \frac{m}{n} \) 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 \( O(k \frac{n}{k} \log \frac{n}{k} + \frac{n}{k} \) time). Hence, this clustering phase takes \( O(n \log n) \) 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 \( O(l \frac{n}{k} \log \frac{n}{k}) \), while search, insertion and deletion cost \( O(\log \frac{n}{k}) \). Finding free time slots makes a pass over the batches in \( O(\frac{n}{k}) \). This is repeated for \( l \) batches, to give an overall time complexity of \( O(\frac{n}{k} \log \frac{n}{k} + \frac{n}{k} \) time). 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 JSC is \( O(n \log n) + O(k \frac{n}{k} m \frac{n}{k} \log \frac{n}{k} + \frac{n}{k} \) time)). However, since the clustering can result in single cluster, \( m \to n \), and the overall complexity of JSC is \( O(n^3) \) in the worst case.
B. The VRC Algorithm

The Vehicle Routing Centric (VRC) algorithm aims to find near-optimal waypoint routing while initially ignoring efficient scheduling of the batch computation. VRC is split into three phases: routing, splitting, and scheduling.

1) Routing Phase: In this phase, VRC builds routes while satisfying the temporal constraint for activities, i.e., for any two consecutive activities \((a_i, a_{i+1})\) in the route, \(t_i + \mathcal{F}(\lambda_i, \lambda_{i+1}) \leq t_{i+1}\). This is done using a modified version of \(k\)-nearest neighbors (k-NN) algorithm, whose solution is then locally optimized using the \(2\text{-OPT}\) heuristic [38].

The modified \(k\)-NN works as follows: Starting from \(\hat{\lambda}\), a route is iteratively built by selecting, from among the \(k\) nearest waypoints which meet the temporal constraint, the one, say, \(\lambda_1\) whose activity has the earliest observation start time. This process resumes from \(\lambda_1\) to find \(\lambda_2\), and so on until there is no feasible neighbor. \(\hat{\lambda}\) 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.

2) Splitting Phase: Say \(R_{i,j} = (\hat{\lambda}, \lambda_i, \ldots, \lambda_j, \hat{\lambda})\) be an energy-infeasible route from the routing phase, which visits \(\lambda_i\) and \(\lambda_j\) as the first and last waypoints from \(\hat{\lambda}\). The goal is to find a suitable waypoint \(\lambda_g\) for \(i \leq g < j\) such that by splitting \(R_{i,j}\) at \(\lambda_g\) and \(\lambda_{g+1}\), 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 \(R_{i,k_1-1}, R_{k_1,k_2-1}, \ldots, R_{k_{g-1},j}\) such that each is a maximal trip and is energy-feasible, i.e., \(E(k_g, k_{g+1} - 1) \leq E\) while \(E(k_g, k_{g+1}) > E\). For each edge \((\lambda_g, \lambda_{g+1}) \in R(k_g, k_{g+1}-1)\), the energy score is the ratio \(\frac{E(k_g, g)}{E(k_g, g + 1)} \leq 1\). A high value indicates that a split at this edge improves the battery utilization.

Utility score. Say \(U(a,b)\) gives the cumulative data capture utility from visiting waypoints in a route \(R_{a,b} \subseteq R_{i,j}\). Say edge \((\lambda_g, \lambda_{g+1}) \in R(k_g, k_{g+1}-1) \subseteq R_{i,j}\) 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 trip at \(\lambda_{g+1}\) and spans until \(\lambda_i\), 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, \(\frac{U(g, l)}{U(k_g, k_{g+1} - 1)}\). 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 \((\lambda_g, \lambda_{g+1}) \in R_{i,j}\), we find the overlap count \(O_g\) as the number of batches from \(\alpha_g\) whose execution slot overlaps with batches from all other activities. The overlap score for edge \((\lambda_g, \lambda_{g+1})\) is given as \(\frac{O_g}{\alpha_g}\). 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 assigned, 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 a valid trip. If either or both of the sub-routes exceed the energy capacity, the splitting phase is recursively applied to that sub-route till all waypoints in the original route are part of some valid trip.

3) 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 JSC.

4) The Core of VRC: The VRC algorithm works as follows (Algorithm 2). After the initial routing phase, energy-uneffable 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.

Algorithm 2: VRC(A, D)

```
1 R ← routing phase
2 for R_{ij} ∈ R do
3   for (λ_g, λ_{g+1}) ∈ R_{ij}, i ≤ g < j do
4     s(g) ← energy score + utility score + compute score
5   R' ← splitting phase based on scores s(i), 1 ≤ i ≤ n
6   for d_j assigned to R_{ij} ∈ R' do
7     apply best scheduling among default assignment and test and swap assignment on R_{ij}
```

5) Time Complexity of VRC: In the routing phase, the modified k-NN takes \(O(kn)\), with \(n\) waypoints and \(k\) number of neighbors. The 2-OPT* algorithm has time complexity \(O(n^4)\). Hence this phase overall has a cost of \(O(n^4)\).

In the splitting phase, calculating the energy score for a route with length \(n\) edges takes \(O(n)\). Calculating the energy score has \(O(n^3)\) complexity, and calculating the compute score has \(O(n)\) complexity. Considering a recursion of length \(n - 1\), the complexity of this phase is \(O(n^4)\).

Combining default assignment and test and swap assignment, VRC’s overall complexity is \(O(n^4)\) in the worst case.

VII. Performance Evaluation

A. Experimental Setup

The OPT solution is implemented using IBM’s CPLEX MILP solver v12 [59]. It uses Python to wrap the objective and constraints, and invokes the parallel solver. Our JSC and VRC heuristics have a sequential implementation using native Python. By default, these scheduling algorithms on our workloads run on an AWS ec2n.4xlarge VM with Intel Xeon
Platinum 8124M CPU, 16 cores, 3.0 GHz, and 42 GB RAM. OPT runs on 16 threads and the heuristics on 1 thread.

We perform real-world benchmarks on flying, hovering, DNN computing and endurance, for a fleet of custom, commercial-grade drones. 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 drone parameters in our analytical experiments.

| $s$ | $e^l$ | $e^h$ | $e^c$ | $E$ |
|-----|-------|-------|-------|-----|
| 4 m/s | 750 J/s | 700 J/s | 20 J/s | 1350 kJ |

B. Workloads

We evaluate the scheduling algorithms for two application workloads: Random (RND) and Depth First Search (DFS). Both have a maximum mission time of 4 h over multiple trips. In the RND workload, $n$ waypoints are randomly placed within a 3.5 km radius from the depot, and with a random activity start time within $[0, 240]$ mins. This is an adversarial scenario with no spatio-temporal locality. The DFS workload is motivated by realistic traffic monitoring needs. We perform a depth-first traversal over a 3.5 km radius of our local city’s road network, centered at the depot. With a $\mathcal{P} = \frac{1}{10}$ probability, we pick a visited vertex as an activity waypoint; $\mathcal{P}$ grows by $\frac{1}{n}$ for every vertex that is not selected, and $n$ are chosen. The start time of these activities monotonically grows.

The table below shows the activity and drone scenarios for each workload. These are based on reasonable operational assumptions and schedule feasibility. We vary the data capture time ($l - t$); batching interval ($\beta$); batch execution time on 2 DNNs ($\rho_M$, $\rho_R$); deadline ($\delta$); utility ($\gamma$); and number of drones ($m$). The load factor $x$ decides the count of activities per mission, $n = x \cdot m$. Drones take at most $r_{\text{max}} = \frac{n}{m}$ trips.

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
l - t & \beta & \rho_M & \rho_R & \delta & \gamma & m & x & n = x \cdot m \\
\hline
[1, 5] & 60 s & 11 s & 98 s & 120 s & [1, 5] & 5, 10, 20, 50 & 2, 4, 8 & 10, \ldots, 200 \\
\hline
\end{array}
\]

For brevity, RNet is only run on DFS. 10 instances of each of these 33 viable workload scenarios are created. We run OPT, JSC and VRC for each to return a schedule and expected utility.

C. Experimental Results

Figures 2a, 2b and 3a show the expected utility per drone for the schedules from the 3 algorithms, for different drone counts and activity load factors. Similarly, Figures 2c, 2d and 3b show the algorithm execution time (log, sec/bs) for them. Each bar is averaged for 10 instances and the standard deviations shown as whiskers. The per drone utility lets us uniformly compare the schedules for different workload scenarios. The total utility – MSP objective function – is the product of the per drone utility shown and the drone count. OPT did not finish (DNF) within 7 h for scenarios with 40 or more activities.

1) OPT offers the highest utility, if it completes executing, followed by VRC, and JSC: Specifically, for the 5-drone scenarios for which OPT completes, it offers an average of 42% higher expected utility than JSC. VRC gives 26% more average utility than JSC for these scenarios, and 75% more for all scenarios they run for. This is as expected for OPT. Since a bulk of the energy is consumed for flying and hovering, VRC, which starts with an energy-efficient route, schedules more activities within the time and energy budget, as compared to JSC. This is evidenced by Figure 4 which reports for MNet the average fraction of activities, which are submitted and successfully scheduled by the algorithms. The remaining activities are not part of any trip. Among all workloads, JSC only schedules 60% of activities, VRC 90%, and OPT 98%. So OPT and VRC are better at packing routes and analytics on the UAVs. OPT and VRC offer more utility for the DFS workload than RND since ≥ 96% of DFS activities are scheduled. They exploit the spatial and temporal locality of activities in DFS.

2) The average flying time per activity in each trip is higher for VRC compared to JSC: Interestingly, at 728 s vs. 688 s per activity, the route-efficient schedules from VRC manage to fly to waypoints farther away from the depot and/or from each other, within the energy constraints, when compared to the schedules from JSC. As a result, it schedules a larger fraction of the activities to gain a higher expected utility.

3) The execution times for VRC and JSC match their time complexity: We use the execution times for JSC to schedule the 300+ workload instances to fit a cubic function in $n$, the number of activities, to match its time complexity of $O(n^3 \cdot l^2)$; since in our runs, $l \in [1, 5]$ and $l \leq n$, we omit that term in the fit. Similarly, we fit a degree-4 polynomial for VRC in $n$. The correlation coefficient for these two fits are high at 0.86 and 0.99, respectively. So, the real-world execution time of our scheduling heuristics match our complexity analysis.

4) OPT is the slowest to execute, followed by VRC and JSC: Despite OPT using 16× more cores than JSC and VRC, its average execution times are > 100 s for just 20 activities. The largest scenario to complete in reasonable time is 40 activities on 5 drones, which took 7 h on average. This is consistent with the NP-hard nature of MSP. As our mission window is 4 h, any algorithm slower than that is not useful.

JSC is fast, and on average completes within 1 s for up to 80 activities. Even for the largest scenario with 50 drones and 200 activities, it takes only 90 s for RND and 112 s for DFS. VRC is slower but feasible for a larger range of activities than OPT. It completes within 3 min for up to 100 activities. But, it takes ≈ 45 min to schedule 200 activities on 50 drones.

5) The choice of a good scheduling algorithm depends on the fleet size and activity count: From these results, we can conclude that OPT is well suited for small drone fleets with about 20 activities scheduled per mission. This completes within minutes and offers about 20% better utility than VRC.
VRC offers a good trade-off between utility and execution time for medium workloads with 100 activities and 50 drones. This too completes within minutes and gives on average about 75% better utility than JSC and schedules over 80% of all submitted activities. For large fleets with 200 or more activities being scheduled, JSC is well suited with fast solutions but has low utility and leaves a majority of activities unscheduled.

6) A higher load factor increases the utility, but causes fewer % of activities to be scheduled: As $x$ increases, we see that the utility derived increases. This is partly due to adequate energy and time being available for the drones to complete more activities in multiple trips. E.g., for the 5-drone case, we use load factors of $x = \{2, 4, 8, 16, 32\}$ for JSC and VRC. There is a consistent growth in the total utility, from 109 to 523 for JSC, and from 121 to 1080 for VRC. There is also a corresponding growth in the number of trips performed per mission, e.g., from 7.5 to 43.2 in total for VRC.

However, the fraction of submitted activities that are scheduled falls. For JSC, its activity scheduled % linearly drops with $x$ from 76% to 23%. But for VRC, the scheduled % stays at about 80% until $x = 8$, at which point the activities saturate the drone fleet’s capacity and the scheduled % falls linearly to 37% for $x = 32$. Interestingly, the utility increases faster than the number of activities scheduled for VRC. This is due to the scheduler favoring activities that offer a higher utility, while avoiding those with a lower utility, causing a 20% increase in utility received per activity between $x = 8$ to $x = 32$.

7) Longer-running edge analytics offer lower on-time utility: We run the same scenarios using RNet and MNet DNNs for the DFS workload. For both JSC and VRC, the data capture utility that accrues from their schedules for the two DNNs is similar. However, since the RNet execution time per batch is
much higher than MNet, there is a drop in on-time utility, by about 32% for both JSC and VRC, due to more deadline violations. As a result, this also causes a drop in total utility for RNet by about 15.9% for JSC and 19% for VRC, relative to MNet. Even for OPT we see a similar trend with a 15.8% drop in the total utility. The runtime of JSC and VRC do not exhibit a significant change between RNet and MNet.

8) Effect of real-world factors: The expected utilities reported above are under ideal conditions. Here, we evaluate their practical efficacy by emulating these schedules using real drone traces to get the effective utility and trip completion rate.

Ideally, each trip generated by JSC and VRC should complete be within a drone’s energy capacity. In practice, factors such as wind or non-linear battery performance can increase or decrease the actual energy consumed. Figure 5 shows the % of scheduled trips that do not complete when using the drone trace. With < 80 activities, all trips complete (not plotted). But but with ≥ 80 activities, some trips in the planned schedule start to fail. At worst, 12% of trips are incomplete in some schedules. So the effect of real-world factors can be significant. Interestingly, for the failed trips, an average 3.6% and a maximum of 7.9% extra battery capacity would allow them to finish the trip. So by maintaining a buffer battery capacity of ≈ 10% when planning a schedule, we can ensure that the drones can complete a trip and return to the depot.

VIII. CONCLUSION AND FUTURE WORK

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 two time-efficient heuristics, JSC and VRC. Evaluations using two workloads, varying drone counts and load factors, and real traces exhibit different trade-offs between utility and execution time. OPT is best for ≤ 20 activities and ≤ 5 drones, VRC for ≤ 100 activities and ≤ 50 drones, and JSC for > 100 activities. 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 onboard 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 also allows us to accept and schedule activities continuously, rather 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.

Acknowledgments. This work is supported by AWS Research Grant, Intelligent Systems Center at Missouri S&T, and NSF grants CCF-1725735 and SCC-1952045. A. Khochare is funded by a Ph.D. fellowship from RBCCPS, IISc, Bangalore. S. K. Das was partially supported by a Satish Dhawan Visiting Chair Professorship at IISc. We thank RBCCPS for access to the drone, and Vishal, Varun and Srikrishna for helping collect the drone traces.
