Pre-print of article to appear in the Proceedings of the *IEEE International Conference on Computer Communications (INFOCOM)*, 2021

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 Computational and Data Sciences, Indian Institute of Science, India
†Department of Computer Science, Missouri University of Science and Technology, USA
Email: *{aakhochare, simmhan}@IISc.ac.in, †{francesco.bettisorbelli, sdas}@mst.edu

***Abstract*—Unmanned Aerial Vehicles (UAVs), or drones, are increasingly used for urban applications like traffic monitoring and construction surveys. Their autonomous navigation allows drones to visit *waypoints* and accomplish *activities* at those locations, 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. Here, we propose a novel *Mission Scheduling Problem (MSP)* for co-scheduling the flight route to visit and record video at waypoints, and their subsequent on-board analysis, for a fleet of drones. The schedule maximizes the utility from the activities, while meeting activity deadlines, and the energy and computing constraints. We first prove that MSP is NP-hard and then offer a mixed integer linear programming (MILP) formulation to optimally solve. Next, we provide two efficient heuristic algorithms, JSC and VRC, to obtain fast, sub-optimal solutions. Our detailed evaluation of these algorithms using real drone benchmarks show the utility-runtime trade-offs of the 3 schedulers 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 expected to enable a wide range of applications in smart cities [1], such as traffic monitoring [2], construction surveys [3], package delivery [4], and even COVID-19 management [5], assisted by the upcoming 5G wireless roll-out [6]. The mobility, agility, and hovering capabilities of drones allow them to rapidly fly to many points of interest (*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 for a certain perdiod using the drone's camera, and analyzing the videos to take decisions, such as changing traffic signaling or sending a patrol car, flagging construction delays, encouraging pedestrians to practise social distancing, etc.

The increasing sophistication of *Deep Neural Networks* (DNNs) and computer vision algorithms enable video analytics to be performed over these recordings for automated decision-making. Typically, these are done after the drone lands, the video is transferred to a cloud or local server, and the DNN inferencing is run. However, certain activities may require low-latency analysis and decisions, as soon as the video is captured at a location. Here, one can leverage the on-board *edge computing* capability available in commercial and bespoke drones to process videos after recording, and immediately report the results to cloud servers over 4G/5G networks. These computing capacities include power-efficient ARM CPUs and NVIDIA Jetson GPUs, designed for edge devices [7].

At the same time, drones are *energy-constrained vehicles* with limited battery capacity, and typical commercial drones can fly for under 1 *hour*. So, the distance between the waypoints to be visited will affect the number of activities that can be completed in one *trip*. Performing edge analytics will consume additional energy, and UAVs also drain energy for hovering and recording video at a location for an activity. So, the energy on the drone should be judiciously managed for the flying, hovering, and computing tasks. Nevertheless, once a drone lands, its exhausted battery can be quickly swapped out for fresh ones and it can start a new trip.

In this paper, we examine how a *UAV fleet operator* in a city can plan *missions* for a captive set of drones to accomplish activities that are periodically provided by users. An *activity* involves visiting a waypoint, hovering and capturing video at that location during 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 that we propose is for the fleet operator to *co-schedule the flight routing among waypoints and the on-board computation for their drones to complete (a subset of) the provided activities, within the energy and compute constraints of the drone, while maximizing the total utility.*

Other earlier works have examined routing of one or more drones for capturing and relaying data to the backend [8], off-loading compute from mobile devices [9], and cooperative video surveillance [10]. There is also literature on scheduling tasks for edge computing that are compute and energy aware, operate on distributed edge resources, and consider deadlines and device reliability [11], [12], [13], [14]. However, none of these examine co-scheduling a fleet of physical drone and the 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)* [15], which generalizes the well known Traveling Salesman Problem (TSP) to find optimal routes given a set of vehicles and customers [16], and the *Job-shop Scheduling Problem (JSP)* [17] for mapping jobs of different execution durations to available resources, which is often used for parallel scheduling of computing tasks to multiprocessors [18].

We make the following specific 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, to maximize 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, that is feasible for small inputs (Section V).
- We also provide two time-efficient heuristic algorithms, JSC and VRC, that can solve 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 solutions, for diverse drone workloads based on real drone benchmarks (Section VII).

In addition, we contrast with related work in Section II, discuss alternative approaches and future work in Section VIII, and offer our conclusions in Section IX.

II. Related Work

This section provides an overview of existing literature on vehicle routing and job-shop scheduling, and how they contrast with MSP and our proposed solutions.

A. Vehicle Routing Problem

The VRP is a TSP variant with multiple salespersons [15] and it has been proven to be NP-hard [19]. This problem has had several extensions to handle realistic delivery scenarios, such as temporal constraints that impose deliveries only at specific time-windows [20], capacity constraints on minimum or maximum vehicle payloads [21], allowing multiple trips for the same vehicle [22], providing profit to vehicles [23], an dealing with traffic congestion [24]. The VRP has also been adapted for planning routes for a fleet of ships [25]. Recently, it has been extended for drone assisted delivery of goods [26].

The authors in [8] consider the problem of scheduling *events* that need to be performed at a specific location, and are submitted to UAVs as they fly. This involves sensing/processing data, and communicating it to the backend over wireless networks. Their dual goal is to minimize both the drone's energy consumption and the operation time. They consider factors such as wind speed and temperature that may affect the route and CPU execution time. However, sensing and processing is a monolithic operation for an event, and not separate but independent tasks like ours that need to be co-scheduled. They minimize the operating time and energy rather than maximize the utility for performing tasks within a time and energy budget, like we do.

Others [9] explore the use of UAVs for off-loading computing from users' mobile devices, and as a data relay between the mobile devices and access points. They consider optimization of the trajectory, bandwidth, and computing of the drone in an iterative manner, for edge computing and relay tasks submitted to a single UAV. They minimize the energy consumption of the drones and the mobile devices, and validate this through simulation for four mobile devices. We instead consider a more practical problem of planning the trajectory and computing for a fleet of drones with possibly hundreds of locations to visit and on-board computing tasks to perform.

Novel architectures have been proposed for energy efficient video surveillance of points of interest (POI) in a city using drones [10]. The UAVs leverage bus rooftops to re-charge themselves, while also being transported to the next POI based on known bus routes. Drones also act as relays for other drones that are capturing video. They formulate the mapping of drones to bus routes as an MILP problem and propose a TSP-based heuristic. Unlike us, scheduling and processing data on-board the drone is not a goal. We do not examine any data off-loading from the drone, nor any piggy-backing mechanisms.

B. Job-shop Scheduling Problem

Scheduling computing tasks on drones is closely aligned with scheduling tasks for edge and fog devices [27], and broadly with parallel workload scheduling [18] and JSC [17].

Dedas [11] proposes an online algorithm for deadline-aware task scheduling for edge computing. This work jointly optimizes networking and computing so as to yield the best possible schedule. It also highlights that workload scheduling on the edge has several dimensions to the problem and jointly optimizing for them, improves the quality of the schedule. Fend et al. [28] propose a framework for cooperative edge computing on autonomous road vehicles, aimed at increasing their computational capabilities in a decentralized manner. Such a system also improves the task execution performance. In [29], the authors combine the optimal placement of data blocks and the optimal scheduling of tasks for reducing the 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.

Several others explore task scheduling where the client is mobile, and off-loads its computing to another nearby edge or fog resource. Typically, these may be categorized based on their mobility models as *predictable* and *unpredictable*. In [30], the mobility of a vehicle is predicted and this is used to select the road-side edge computing unit to which the compute is off-loaded. In [31], the authors take an extreme view and assume that mobile edge devices interact with other such devices intermittently and at random. This makes it challenging to determine if tasks should be off-loaded to another proximate mobile edge device for reliable completion. The problem we solve is complementary, with the possible waypoints known ahead, and we perform predictable UAV route planning and schedule the computing locally on the edge.

Scheduling on energy-constrained edge has also drawn attention [12], where an energy-aware off-loading scheme is proposed for jointly optimizing communication and computation resource allocation on the edge and to limit latency. In our problem, we also consider the energy for the drone flight and try to 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. Figure 1 illustrates a sample MSP scenario.

A. *UAV System Model*

Let $\widehat{\lambda} = (0, 0, 0)$ be the *location* of a UAV depot in the city (see Figure 1, left) centered at the origin of a 3D Cartesian coordinate system [1].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 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 Cartesian space; 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. For simplicity, we ignore the energy for video capture since it is negligible in practice. Also, a drone that returns to the depot can get a fresh battery immediately to start a new trip.

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 subsequently processing all of the captured data is κ_i floating point operations [2]. 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 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 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 $q_i = |B_i| = \lceil \frac{\bar{t}_i - t_i}{\beta} \rceil$. The computational cost to process each batch is $\kappa_i^k = \frac{\kappa_i}{q_i}$ floating-point operations, 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 $\rho_i^k = \lceil \kappa_i^k \cdot \frac{1}{\pi} \rceil$; for simplicity, we discretize all time-units into integers.

We make some simplifying assumptions. Only one batch may be executed at a time on-board a drone and it should run to completion before scheduling another. There is no concurrency, pre-emption, or checkpointing. 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 should be executed in sequence, i.e., complete processing b_i^k before processing b_i^{k+1}.

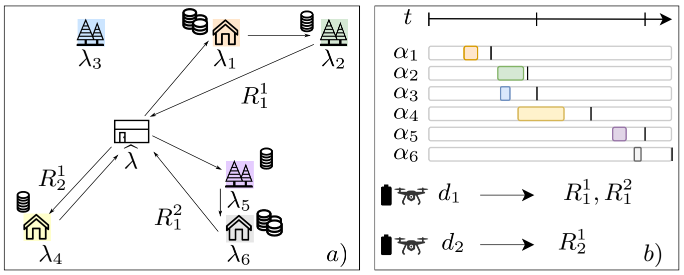


Fig. 1. Sample MSP scenario. a) shows a city with the depot ($\widehat{\lambda}$); 6 waypoints to visit (λ_i) with some utility; and possible trip routes for drones (R_j^i). b) has the corresponding 6 activities (α_i) with data capture duration (shaded) and compute deadline (vertical line) and the two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

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 (γ_i)* accrued by a drone for an activity α_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 caputed 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 by 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.

IV. Problem Formulation

The Mission Scheduling Problem (MSP) can be summarized as: *Given a UAV depot in a city with a fleet of captive drones, and a set of observe and compute 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 compute onto 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 from users. Periodically, a mission schedule is planned to serve

[1]Since the route planning is in a city, without loss of generality (WLOG) and for simplicity, we use Cartesian coordinates for 3D positioning rather a geo-spatial system such as latitude, longitude, and elevation, or UTM.

[2]We assume that the computing performed is numerical and floating-point heavy, as is often the case with Deep Neural Network model inference.

some or all these activities using all the drones in the 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 do 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 $A^r_j = (\alpha^r_{j_1}, \dots, \alpha^r_{j_n}) \subseteq A$ where $\alpha^r_{j_x} \in A$, $j_n \le n$, and no activity appears twice within a mission. Further, we have $\alpha^r_{j_x} \prec \alpha^r_{j_{x+1}}$, i.e., the observation start and end times of an activity in the mission sequence fully precede those of the next activity in it, $\bar{t}^r_{j_x} \le t^r_{j_{x+1}}$. Also, $A^x_j \cap A^y_k = \varnothing \; \forall j,k,x,y$ 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, i.e., $\sum_j \sum_r |A^r_j| \le n$.

The *route* for the r^{th} trip of drone d_j is given by $R^r_j = (\widehat{\lambda}, \lambda^r_{j_1}, \dots, \lambda^r_{j_n}, \widehat{\lambda})$, 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 $\lambda^r_{j_k}$ for the activity $\alpha^r_{j_k}$ in the mission sequence. For uniformity, we denote the first and the last depot location in the route as $\lambda^r_{j_0}$ and $\lambda^r_{j_{n+1}}$, respectively. Clearly, $|R^r_j| = j_n + 2$.

A drone d_j, given the r^{th} trip of its route R^r_j, it starts at the depot, visits each waypoint in the sequence, and returns to the depot, where it may get a fresh battery and start on the $(r+1)^{th}$ route. Let drone d_j leave a waypoint location in its route, $\lambda^r_{j_i}$, at *departure time* $\tau^r_{j_i}$ and reach the next waypoint location, $\lambda^r_{j_{i+1}}$, at *arrival time* $\bar{\tau}^r_{j_{i+1}}$. Let the function $\mathcal{F}(\lambda_p, \lambda_q)$ give the *flying time* between λ_i and λ_j. Since the drone has a constant flying speed, we have $\bar{\tau}^r_{j_{i+1}} = \tau^r_{j_i} + \mathcal{F}(\lambda^r_{j_i}, \lambda^r_{j_{i+1}})$.

The drone must hover at each waypoint $\lambda^r_{j_i}$ between t^r_j and $\bar{t}^r_j$ while recording the video, and it departs the waypoint after this, i.e., $\tau^r_{j_i} = \bar{t}^r_{j_i}$. If the drone arrives at this waypoint at time $\bar{\tau}^r_{j_i}$, that is before the observation start time t_j, it *hovers* here for a duration of $t^r_j - \bar{\tau}^r_{j_i}$, and then continues hovering during the activity's video capture. If a drone arrives at $\lambda^r_{j_i}$ after t^r_j, it is invalid since the video capture for the activity cannot be done for the whole duration. So, $\bar{\tau}^r_{j_i} \le t^r_{j_i} \le \tau^r_{j_i}$. Also, since the deadline for on-time computation over the captured data is $\delta^r_{j_i}$, we require $\delta^r_{j_i} \ge \bar{t}^r_{j_i}$.

Once the drone finishes capturing video for the last activity in its r^{th} trip, it returns back to the depot location at time $\bar{\tau}^r_{j_{n+1}} = \tau^r_{j_n} + \mathcal{F}(\lambda^r_{j_n}, \widehat{\lambda})$. Hence, the *total flying time* for a drone d_j for its r^{th} trip is:

$$f^r_j = \sum_{i=0}^{n} (\bar{\tau}^r_{j_{i+1}} - \tau^r_{j_i})$$

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

$$h^r_j = \sum_{i=1}^{n} (t^r_{j_i} - \bar{\tau}^r_{j_i}) + \sum_{i=1}^{n} (\bar{t}^r_{j_i} - t^r_{j_i}) = \sum_{i=1}^{n} (\bar{t}^r_{j_i} - \bar{\tau}^r_{j_i})$$

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

Let the scheduler assign the *time slot* $[\theta^k_{j_i}, \bar{\theta}^k_{j_i})$ for executing a batch $b^k_{j_i}$ of activity α_{j_i} on drone d_j, where $\bar{\theta}^k_{j_i} = \theta^k_{j_i} + \rho^k_i$, based on the batch execution time. We define a completion function for each activity α_{j_i}, for the three utility values:

- The *data capture completion* $u_{j_i} \in \{0, 1\}$. The value is 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* $0.0 \le \bar{u}_{j_i} \le 1.0$ indicates the fraction of batches of that activity that are completed on-board the drone. Let $\bar{\mu}^k_i = 1$ if the batch b^k_i of activity α_i is completed on-board, and $\bar{\mu}^k_i = 0$ if it is not completed on-board the drone. Then, $\bar{u}_{j_i} = \frac{\sum_k \bar{\mu}^k_i}{q_i}$.
- The *on-time completion* $0.0 \le \bar{\bar{u}}_{j_i} \le 1.0$ indicates the fraction of batches of that activity that are fully completed within the deadline. Similarly, let $\bar{\bar{\mu}}^k_i = 1$ if the batch b^k_i of activity α_i is completed on-time, i.e., $\bar{\theta}^k_i \le \delta_i$, and $\bar{\bar{\mu}}^k_i = 0$ otherwise. So, $\bar{\bar{u}}_{j_i} = \frac{\sum_k \bar{\bar{\mu}}^k_i}{q_i}$.

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

$$c_j = \sum_{\alpha_i \in A} (\bar{\mu}^k_{j_i} + \bar{\bar{\mu}}^k_{j_i}) \cdot \rho^k_i$$

B. Optimization of MSP

Based on these, the **objective** of the optimization is:

$$\arg\max \sum_{\alpha_i \in A} U_i$$

i.e., assign drones to activity waypoints and activity batches to drone for computing slots to maximize the utility derived 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:

$$(t_{j_i} + k \cdot \beta) \le \theta^k_{j_i} \qquad \bar{\theta}^k_{j_i} \le \theta^{k+1}_{j_i} \qquad \bar{\theta}^k_i \le \bar{\tau}_{j_{n+1}}$$

i.e., the data capture for a duration of β for the k^{th} batch of the activity is complete 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 $\forall [\theta^x_{j_p}, \bar{\theta}^x_{j_p})$ and $[\theta^y_{j_q}, \bar{\theta}^y_{j_q})$ slots assigned to batches b^x_p and b^y_q on drone d_j, we have $[\theta^x_{j_p}, \bar{\theta}^x_{j_p}) \cap [\theta^y_{j_q}, \bar{\theta}^y_{j_q}) = \varnothing$.

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

$$E^r_j = f^r_j \epsilon^f + h^r_j \epsilon^h + c^r_j \epsilon^c \le E$$

V. Optimal Solution for MSP

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

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 [19]. In addition, MSP considers multiple-trips, time-windows, energy-constraints, and utilities.

TABLE I
CONSTRAINTS FOR OPT MILP FORMULATION.

C.	Expression	Meaning
1	$\sum_{k\in\mathcal{D}}\sum_{l\in\mathcal{R}}\sum_{j\in\overrightarrow{i}} x_{ij}^{kl} \le 1, \quad \forall i \in \mathcal{V}'$	The waypoint for an activity α_i is visited only once.
2	$\sum_{j\in\overrightarrow{0}} x_{0j}^{kl} - \sum_{j\in\overleftarrow{0}} x_{j0}^{kl} = 0, \quad \forall k \in \mathcal{D}, l \in \mathcal{R}$	A drone trip l starting from the depot must also end there.
3	$\sum_{j\in\overrightarrow{0}} x_{0j}^{kl} = 1 \iff \sum_{j\in\overrightarrow{i}} x_{ij}^{kl} = 1, \quad \forall i \in \mathcal{V}', k \in \mathcal{D}, l \in \mathcal{R}$	A drone k must visit at least one waypoint on each trip l.
4	$\sum_{i\in\overleftarrow{j}} x_{ij}^{kl} - \sum_{i\in\overrightarrow{j}} x_{ji}^{kl} = 0, \quad \forall k \in \mathcal{D}, j \in \mathcal{V}', l \in \mathcal{R}$	A drone k visiting waypoint j must also fly out from there.
5	$(t_j - \mathcal{F}_{0j}) \cdot \sum_{k\in\mathcal{D}}\sum_{l\in\mathcal{R}} x_{0j}^{kl} \ge 0, \quad \forall j \in \mathcal{V}'$	Any drone flying to waypoint j from the depot must reach before its observation start time t_j.
6	$(t_j - \bar{t}_i - \mathcal{F}_{ij}) \cdot \sum_{k\in\mathcal{D}}\sum_{l\in\mathcal{R}} x_{ij}^{kl} \ge 0, \quad \forall i \in \mathcal{V}', j \in \overrightarrow{i}$	Any drone flying to waypoint j from i must reach before its observation start time t_j.
7	$\bar{\tau}_{k_{n+1}}^{l} = \sum_{i\in\mathcal{V}'} x_{i0}^{kl} \cdot (\bar{t}_i + \mathcal{F}_{i0}), \quad \forall k \in \mathcal{D}, l \in \mathcal{R}$	Decides the landing time of drone k at the depot after trip l.
8	$\bar{\tau}_{k_{n+1}}^{l} \le \tau_{\max}, \quad \forall k \in \mathcal{D}, l \in \mathcal{R}$	Depot landing times for all trips is within the maximum time.
9	$t_i + (g+1) \cdot \beta \le \theta_i^g, \quad \forall i \in \mathcal{V}', g \in \mathcal{B}_i$	Batch g of activity α_i must be observed before it is processed.
10	$\bar{\theta}_i^g < \theta_i^{g+1}, \quad \forall i \in \mathcal{V}', g \in \mathcal{B}_i$	Processing of batch g of activity α_i must precede batch $g+1$.
11	$\sum_{j\in\overrightarrow{i}} x_{ij}^{kl} + \sum_{b\in\overrightarrow{a}} x_{ab}^{kl} - 1 \le w_{ia}^{gh} + w_{ai}^{hg}, \quad \forall i, a \in \mathcal{V}', i < a, g \in \mathcal{B}_i, h \in \mathcal{B}_a, k \in \mathcal{D}, l \in \mathcal{R}$	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 [17].
12	$\bar{\theta}_i^g - \theta_a^h \le M \cdot (1 - w_{ia}^{gh}), \quad \forall i, a \in \mathcal{V}, i \ne a, g \in \mathcal{B}_i, h \in \mathcal{B}_a$	
13	$y_{ik}^{lg} = 1 \Rightarrow \bar{\theta}_i^g + M\left(1 - \sum_{j\in\overrightarrow{i}} x_{ij}^{kl}\right) \le \delta_i, \quad \forall i \in \mathcal{V}', g \in \mathcal{B}_i, k \in \mathcal{D}, l \in \mathcal{R}$	Decision variable for batches that complete before deadline.
14	$z_{ik}^{lg} = 1 \Rightarrow \bar{\theta}_i^g + M\left(1 - \sum_{j\in\overrightarrow{i}} x_{ij}^{kl}\right) \le \bar{\tau}_{k_{n+1}}^{l}, \forall i \in \mathcal{V}', g \in \mathcal{B}_i, k \in \mathcal{D}, l \in \mathcal{R}$	Decision variable for batches that complete before landing.
15	$\sum_{i\in\mathcal{V}}\left(\sum_{j\in\overrightarrow{i}}\left(x_{ij}^{kl} \cdot \mathcal{F}_{ij} \cdot \epsilon^f\right) + \sum_{g\in\mathcal{B}_i}\left(z_{ik}^{lg} \cdot \kappa_i^g \cdot \epsilon^c\right) + \sum_{j\in\overrightarrow{i}}\left(x_{ij}^{kl} \cdot (\bar{t}_j - (\bar{t}_i + \mathcal{F}_{ij})) \cdot \epsilon^h\right)\right) \le E, \quad \forall k \in \mathcal{D}, l \in \mathcal{R}$	Sum of energy consumed for flying, hovering and computing on trip l of drone k should be within battery capacity.

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 $T_h = \sum_{e\in\mathcal{E}} \mathcal{F}(e)$, where $\mathcal{E}$ is the set of edges and $\mathcal{F}(e)$ is the flying time for traversing an edge [33], 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 $\bar{t}_i = +\infty$ [16]. Clearly, a VRP variant with energy-constrained vehicles is still NP-hard, by just relaxing those constraint 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 [22] 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 [34]. So any variant which introduces constraints is again NP-hard by a simple reduction to relax those constraint to JSP.

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

B. *The* OPT *Algorithm*

The OPTIMAL MISSION SCHEDULER (OPT) algorithm offers an optimal solution to MSP by modelling it as a multi-commodity flow problem (MCF), similar to [35], [10]. We reformulate the MSP definition as an MILP formulation.

The paths in the city are modelled as a *complete graph*, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, between the n activity 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 waypoint vertices. 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 do. Let $\mathcal{R} = \{1, \ldots, r_{\max}\}$ be the 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 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, 0 otherwise [17].

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 on-board completion be $\bar{\Gamma}_i = \frac{\bar{\gamma}_i}{q_i}$, and on-time completion be $\bar{\bar{\Gamma}}_i = \frac{\bar{\bar{\gamma}}_i}{q_i}$, for activity α_i. Finally, let M be a sufficiently large constant.

Using these, the MILP objective is:

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

subject to the constraints in Table I.

VI. HEURISTIC ALGORITHMS FOR MSP

Since MSP is NP-hard, OPT can be reasonably performed 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 batches scheduling while ignoring optimizations of routing in terms of energy. JSC is split into two phases: *clustering* and *scheduling*.

1) The Clustering Phase: First, we use the ST-DBSCAN algorithm [36] to do time-efficient spatio-temporal clustering 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. For each cluster $C_i \in \mathbb{C}$, 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; analogously, 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 respect to T_i^L. Recalling that there are m drones available at $\widehat{t} = 0$, 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_1^L and released at time T_1^U; $c_2 = \frac{m-c_1}{n} \cdot |C_2|$ allocated to C_2 from T_2^L to T_2^U (assuming $T_2^L < T_1^U$), and so on.

2) The Scheduling Phase: Here, the activities are assigned to drones. 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 requirements. Assignments are done in two steps: *default assignment* 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 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 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 $P_i^+ = \bigcup_i P_{b_i^k}$ 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.

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 deadline violation, the *test and swap assignment* performed, and the best scheduling is applied.

4) Time Complexity of JSC: The ST-DBSCAN time complexity is $\mathcal{O}(n \log n)$, where n is the number of waypoints. As opposed to k-means algorithm, ST-DBSCAN does not require one to specify the number of clusters *a priori*. Let k be the number of clusters formed, with $\approx \frac{n}{k}$ waypoints each. For k times, we compute the the min-max of sets of size $\frac{n}{k}$, sort the k elements, and finally make $\frac{n}{k}$ assignments. So this drones-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.

Algorithm 1: JSC(A, D)

```
1 $\mathbb{C} \leftarrow$ clustering phase
2 for $C_k \in \mathbb{C}$ do
3     for $\alpha_i \in C_k$ do
4         for $d_j$ assigned to $C_k$ do
5             if $\alpha_i \cup A_j$ is feasible then
6                 apply best scheduling among default and
                  test and swap assignment on $\widehat{B_i}$
```

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 JSC 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 single cluster, $m \to n$, and the *overall complexity* of JSC is $\mathcal{O}(n^3 l^2)$ in the worst case.

B. The VRC Algorithm

The VEHICLE ROUTING CENTRIC (VRC) algorithm aims to find near-optimal waypoint routing while initially ignoring the optimizations for scheduling batch computation. VRC is split into three phases: *routing*, *splitting*, and *scheduling*.

1) The Routing Phase: In this phase, VRC 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 [37].

The modified k-NN works 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 [16]. However, routes found here may be infeasible for a drone to complete within its energy constraints.

2) The 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 and λ_j as the first and last waypoints from $\widehat{\lambda}$. The goal is to find a suitable waypoint $\lambda_g, i \leq g < j$ such that by splitting $R_{i,j}$ at λ_g, 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 and is energy-feasible, i.e., $E(k_y, k_{y+1} - 1) \leq E$ while $E(k_y, k_{y+1}) > E$. For each edge $(\lambda_g, \lambda_{g+1}) \in R_{(k_y,k_{y+1}-1)}$, the *energy score* is the ratio $\frac{E(k_y,g)}{E} \leq 1$. A high value indicates that a split at this edge 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 $(\lambda_g, \lambda_{g+1}) \in R_{(k_y,k_{y+1}-1)} \subseteq R_{i,j}$ 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 of 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 a sequential partitioning of the route done earlier.

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 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) The 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-unfeasible routes are split in feasible ones in the *splitting phase*, and then drones are allocated to them. Finally, in the *scheduling phase* is applied the best scheduling between the *default assignment* and the *test and swap assignment*.

5) Time Complexity of VRC*:* In the *routing phase*, the modified k-NN takes $\mathcal{O}(kn)$, where k denotes the number of neighbors, while the 2-OPT* algorithm has time complexity $\mathcal{O}(n^4)$, hence this phase costs overall $\mathcal{O}(n^4)$.

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 $\mathcal{O}(n^3)$

Algorithm 2: VRC(A, D)

```
1 $\mathbb{R} \leftarrow$ routing phase
2 for $R_{ij} \in \mathbb{R}$ do
3     for $(\lambda_g, \lambda_{g+1}) \in R_{ij}, i \leq g < j$ do
4         $s(g) \leftarrow$ energy score + utility score + compute score
5 $\mathbb{R}' \leftarrow$ splitting phase based on scores $s(i), 1 \leq i \leq n$
6 for $d_j$ assigned to $R_{ij} \in \mathbb{R}'$ do
7     apply best scheduling among default assignment and test
        and swap assignment on $R_{ij}$
```

When combined with the complexities of *default assignment* and *test and swap assignment*, the *overall complexity* of VRC is $\mathcal{O}(n^4)$ in the worst case.

VII. Performance Evaluation

A. Experimental Setup

The OPT solution is implemented using IBM's CPLEX MILP solver v12 [38]. 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 c5n.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.

We perform 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 ≈ 30 min with a range of 3.5 km. Based on our benchmarks, we use the following drone parameters in our analytical 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 *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 $\frac{1}{10}$ 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 ($\bar{t} - t$); batching interval (β); batch execution time on 2 DNNs (ρ_M, ρ_R)[3]; deadline (δ); utility (γ); and number of drones (m). The *load factor* x decides the count of activities per mission, $n = x \cdot m$. Drones take at most $r_{\max} = \frac{n}{m}$ trips.

$\bar{t} - t$	β	ρ_M	ρ_R	δ	γ	m	x	$n = x \cdot m$
$[1, 5]$	60 s	11 s	98 s	120 s	$[1, 5]$	5, 10, 20, 50	2, 4, 8	10, …, 200

For brevity, RNet is only run on DFS. 10 *instances* of each of these 33 viable workload scenarios are generated. We run OPT, JSC, and VRC for each to return a schedule. Their results are analyzed and presented next.

C. Experimental Results

Figures 2a, 2b, and 3a show the *utility per drone* given by schedules from the three algorithms, for different drone counts and activity load factors. Similarly, Figures 2c, 2d, and 3b show the *algorithm execution time (log, secs)* 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% more 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.

[3]We run *SSD Mobilenet v2 DNN* (MNet, ρ_M) [39], popular for analyzing drone footage [40], and *FCN Resnet18 DNN* (RNet, ρ_R) [41] on the TX2

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 $\geq 96\%$ 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 more utility.

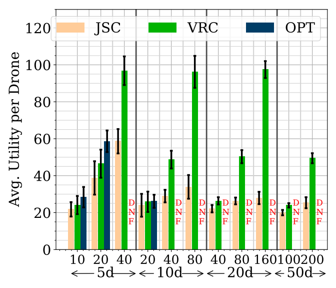
3) The execution times for VRC *and* JSC *match their time complexity:* We use the execution times for JSC to schedule the 300+ workload instances to fit a *cubic function* in n, the number of activities, to match its time complexity of $\mathcal{O}(n^3 \cdot l^2)$; since in our runs, $l \in [1, 5]$ 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.

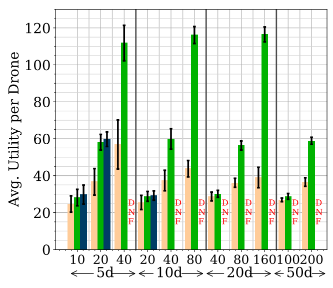
4) OPT *is the slowest to execute, followed by* VRC *and* JSC*:* Despite OPT using 16× more cores than JSC and VRC, its average execution times are > 100 s for just 20 activities. The largest scenario we are able to run within reason is 40 activities on 5 drones, which took 7 h on average. This is consistent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

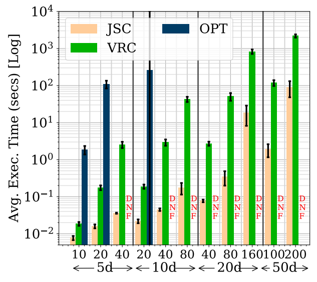
JSC is fast, and on average completes within 1 s for up to 80 activities. Even for the largest scenario with 50 drones and 200 activities, it takes only 90 s for RND and 112 s for DFS. VRC is slower but feasible for a larger range of activities than OPT. It completes within 3 min for up to 100 activities.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


(a) Utility per drone, RND.


(b) Utility per drone, DFS.


(c) Alg. runtime, RND.

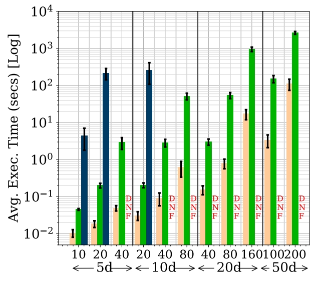
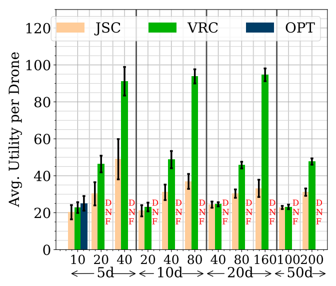

(d) Alg. runtime, DFS.

Fig. 2. *Utility per drone* and *algorithm runtime* of the three MSP algorithms, for the RND and DFS workloads on MNet. On the X axis, the number of drones (outer) and activities per drone (inner) increase. OPT is solved on $16\times$ cores while JSC and VRC run on just 1. DNF indicates OPT did not finish.


(a) Utility per drone.

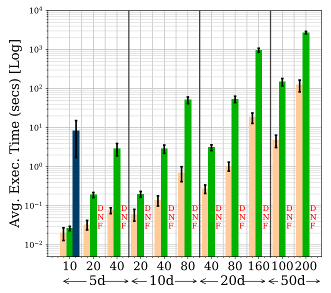
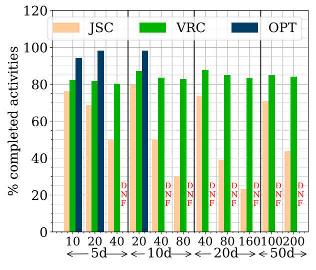

(b) Alg. runtime.

Fig. 3. *Utility per drone* and *algorithm runtime* for RNet on DFS.


(a) RND.

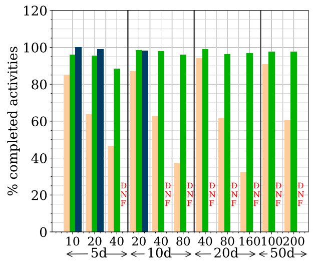

(b) DFS.

Fig. 4. Fraction (%) of submitted activities scheduled per mission for MNet.

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 DNN 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 runtimes of JSC and VRC do not exhibit a significant change between RNet and MNet.

VIII. Discussion and Future Work

The MSP that we have proposed is just one variant of an entire class of fleet co-scheduling problems for drones. Other architectures to be explored in future include consideration of 4G/5G network coverage and bandwidth to send edge results to the backend, or even off-load the captured data to the cloud for computing if it is infeasible on the drone. There will be energy and latency trade-offs. Even the routing can be aware of the cellular coverage to ensure such off-loading can happen on a trip, deterministically. We can also pre-provision and schedule cloud resources for tasks that did not completed on-board the drone and were off-loaded at the depot.

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 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.

Lastly, we can examine the impact of runtime conditions in rescheduling. E.g., changing winds may affect the energy use, while changing application needs may make some scheduled waypoints unnecessary or require us to increase their data capture times. It would be interesting to study online scheduling to complement batch scheduling, where additional activities can be submitted when a drone is on a scheduled trip.

IX. Conclusion

In this paper, we have introduced a novel Mission Scheduling Problem (MSP) to co-schedule routes and analytics for a fleet of homogeneous drones, for maximizing the utility obtained from completing activities. We propose an optimal mixed integer linear programming (MILP) implementation, OPT, and design two time-efficient heuristic algorithms, JSC and VRC, to solve MSP. Our detailed evaluation for two workloads, varying drone counts and load factors shows that these offer different trade-offs on utility and execution time, with OPT best for ≤ 20 activities and ≤ 5 drones, VRC for ≤ 100 activities and ≤ 50 drones, and JSC for > 100 activities. Their time complexity matches reality. We also see that improving the load factor favors VRC, with a higher utility and good activity scheduling rate. The schedules work well for fast and slow DNNs, though on-time utility drops for the latter.

REFERENCES

[1] F. Mohammed, A. Idries, N. Mohamed, J. Al-Jaroodi, and I. Jawhar, "UAVs for smart cities: Opportunities and challenges," in *IEEE International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 267–273, 2014.

[2] K. Kanistras, G. Martins, M. J. Rutherford, and K. P. Valavanis, "A survey of unmanned aerial vehicles (uavs) for traffic monitoring," in *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 221–234, IEEE, 2013.

[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 *International Workshop on Mobile Computing Systems and Applications*, pp. 3–8, 2019.

[4] K. Dorling, J. Heinrichs, G. G. Messier, and S. Magierowski, "Vehicle routing problems for drone delivery," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 1, pp. 70–85, 2016.

[5] D. G. Costa and J. P. J. Peixoto, "Covid-19 pandemic: a review of smart cities initiatives to face new outbreaks," *IET Smart Cities*, 2020.

[6] 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 Journal on Selected Areas in Communications*, vol. 36, no. 11, pp. 2486–2496, 2018.

[7] S. Jung, S. Hwang, H. Shin, and D. H. Shim, "Perception, guidance, and navigation for indoor autonomous drone racing using deep learning," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2539–2544, 2018.

[8] N. H. Motlagh, M. Bagaa, and T. Taleb, "Energy and delay aware task assignment mechanism for uav-based iot platform," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6523–6536, 2019.

[9] X. Hu, K.-K. Wong, K. Yang, and Z. Zheng, "Uav-assisted relaying and edge computing: Scheduling and trajectory optimization," *IEEE Transactions on Wireless Communications*, vol. 18, no. 10, pp. 4738–4752, 2019.

[10] 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 *IEEE International Conference on Computer Communications (INFOCOM)*, pp. 1043–1051, 2018.

[11] 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 *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 2287–2295, IEEE, 2019.

[12] J. Zhang, X. Hu, Z. Ning, E. C.-H. Ngai, L. Zhou, J. Wei, J. Cheng, and B. Hu, "Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2633–2645, 2017.

[13] Y. Zhang, X. Chen, Y. Chen, Z. Li, and J. Huang, "Cost efficient scheduling for delay-sensitive tasks in edge computing system," in *IEEE International Conference on Services Computing (SCC)*, pp. 73–80, 2018.

[14] Y. Mao, J. Zhang, and K. B. Letaief, "Joint task offloading scheduling and transmit power allocation for mobile-edge computing systems," in *IEEE wireless communications and networking conference (WCNC)*, pp. 1–6, 2017.

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

[16] P. Toth and D. Vigo, *The vehicle routing problem.* SIAM, 2002.

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

[18] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Computing Surveys (CSUR)*, vol. 31, no. 4, pp. 406–471, 1999.

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

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

[21] E. Uchoa, D. Pecin, A. Pessoa, M. Poggi, T. Vidal, and A. Subramanian, "New benchmark instances for the capacitated vehicle routing problem," *European Journal of Operational Research*, vol. 257, no. 3, pp. 845–858, 2017.

[22] D. Cattaruzza, N. Absi, and D. Feillet, "Vehicle routing problems with multiple trips," *Annals of Operations Research*, vol. 271, p. 127159, 2018.

[23] F. Stavropoulou, P. P. Repoussis, and C. D. Tarantilis, "The vehicle routing problem with profits and consistency constraints," *European Journal of Operational Research*, vol. 274, no. 1, pp. 340–356, 2019.

[24] 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 *IFIP International Conference on Advances in Production Management Systems*, pp. 190–197, Springer, 2018.

[25] K. Fagerholt, "Optimal fleet design in a ship routing problem," *International transactions in operational research*, vol. 6, no. 5, pp. 453–464, 1999.

[26] 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, 2019.

[27] P. Varshney and Y. Simmhan, "Characterizing application scheduling on edge, fog, and cloud computing resources," *Software: Practice and Experience*, vol. 50, no. 5, pp. 558–595, 2020.

[28] J. Feng, Z. Liu, C. Wu, and Y. Ji, "Mobile edge computing for the internet of vehicles: Offloading framework and job scheduling," *IEEE vehicular technology magazine*, vol. 14, no. 1, pp. 28–36, 2018.

[29] C. Li, J. Bai, and J. Tang, "Joint optimization of data placement and scheduling for improving user experience in edge computing," *Journal of Parallel and Distributed Computing*, vol. 125, pp. 93–105, 2019.

[30] Z. Ning, J. Huang, X. Wang, J. J. Rodrigues, and L. Guo, "Mobile edge computing-enabled internet of vehicles: Toward energy-efficient scheduling," *IEEE Network*, vol. 33, no. 5, pp. 198–205, 2019.

[31] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura, "Serendipity: Enabling remote computing among intermittently connected mobile devices," in *ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pp. 145–154, 2012.

[32] 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.

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

[34] R. L. Graham, "Bounds for certain multiprocessing anomalies," *Bell system technical journal*, vol. 45, no. 9, pp. 1563–1581, 1966.

[35] B. Zmazek, "Multiple trips within working day in capacitated vrp with time windows," in *WSEAS international conference on Computers (ICCOMP)*, pp. 93–99, 2006.

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

[37] J.-Y. Potvin and J.-M. Rousseau, "An exchange heuristic for routeing problems with time windows," *Journal of the Operational Research Society*, vol. 46, no. 12, pp. 1433–1446, 1995.

[38] IBM ILOG, "V12.1: Users manual for cplex," tech. rep., International Business Machines Corporation, 2009.

[39] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[40] J. Wang, Z. Feng, Z. Chen, S. George, M. Bala, P. Pillai, S.-W. Yang, and M. Satyanarayanan, "Bandwidth-efficient live video analytics for drones via edge computing," in *IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 159–173, 2018.

[41] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3431–3440, 2015.