A Multi-UAV Router and Scheduler for Executing Spatially Scattered Real-Time Tasks

Sreyashi Mukherjee
ATDC, IIT Kharagpur, India

Sachin Yadav
Civil Engineering, IIT Kharagpur, India

Yedla Anil Kumar
Chemical Engineering, IIT Kharagpur, India

Arnab Sarkar
ATDC, IIT Kharagpur, India

Abstract

Cyber-Physical Systems (CPSs) operating in remote or field scenarios often face limited local processing capacity, necessitating complex real-time monitoring and control via remote processing through mobile edge networks, satellite systems, or UAVs. With recent advancements, UAVs are increasingly being favored for such applications, particularly in isolated areas beyond edge or satellite network coverage. This paper presents a unified UAV scheduling and routing framework for executing geographically distributed real-time CPS tasks under both periodic and aperiodic arrival models. We address the challenge of minimizing the number of UAVs required while ensuring strict adherence to task deadlines across diverse temporal and spatial settings. At first, we propose an efficient heuristic strategy called *UAV Scheduling and Routing Algorithm for Real-time Tasks - Periodic Arrivals* ($USRART-P$), which decomposes applications into task instances and sequentially creates per-UAV routes and schedules within a hyperperiod, maximizing the number of task instances each UAV can cover while meeting deadlines. Adapting to this framework, we develop two additional variants to handle aperiodic CPS tasks: $USRART-SA$ *for Synchronous Aperiodic Arrivals* (common arrival time, distinct deadlines) and $USRART-AA$ *for Asynchronous Aperiodic Arrivals* (distinct but known arrival times and deadlines). For the case of periodic tasks, we frame the problem as a constraint optimization formulation which aims to minimize the number of UAVs that are required to generate static hyperperiodic travel routes with task execution schedules for all UAVs, and discuss how the formulation can be adapted for aperiodic tasks. Solution to this formulation using standard off-the-shelf solvers achieves optimality but incurs high computational overheads. Through extensive simulations, we show that $USRART$ exhibits high performance across diverse operational scenarios, varying task distributions, execution demands, and spatial layouts. The results emphasize $USRART$'s flexibility and effectiveness in real-world UAV-based CPS scenarios, especially in environments with limited resources and infrastructure.

2012 ACM Subject Classification Computer systems organization → Real-time system specification; Computer systems organization → Real-time system architecture; Computer systems organization → Robotics

Keywords and phrases UAV Scheduling, Task Allocation, Optimization, Execution Time

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2025.4

1 Introduction

1.1 Proliferation of Real-time IoT Applications

Recent advancements in computing and communication technologies have led to a surge in Internet of Things (IoT) applications across various domains such as smart cities, industrial automation, healthcare, smart agriculture etc,. These IoT applications often require continu-

ous control, sensing, and actuation, with strict demands on timeliness and reliability. For example, IoT-enabled smart agricultural systems monitor soil conditions and crop health in real-time, allowing for automated irrigation and fertilization based on live data feeds [30]. Similarly, predictive maintenance in industrial IoT (IIoT) leverages real-time data to monitor equipment performance, preventing unexpected breakdowns and optimizing operational efficiency [17]. In traffic management, IoT systems can monitor vehicle flow and congestion, detect road hazards, air quality etc. to perform real-time adjustments in traffic signals for improving road safety and reduce delays [1]. As the number and complexity of these IoT applications continue to grow, there is an increasing need for flexible/scalable computation and communication solutions in order to support the high bandwidth, low-latency processing and connectivity demands.

1.2 UAV-Based Execution of Real-time IoT Applications

Traditionally, computation, sensing and communication needs of IoT applications have been met using local on-board facilities. However in recent times, the limited processing facilities available on-board often become overwhelmed as the applications become progressively smarter, distributed, heterogeneous and complex in nature. To address this challenge, IoT systems have sometimes resorted to remote computation on nearby edge servers. Equipped with high bandwidth communication, such edge-based systems can become an attractive alternative mechanism for executing real-time IoT applications in diverse domains like agriculture, transportation, emergency response etc. However in many scenarios, such as remote or rapidly changing environments, fixed edge infrastructure may not be readily available.

To alleviate this problem, mobile edge computing implemented with Unmanned Aerial Vehicles (UAVs), are being envisaged as vital enablers for executing real-time IoT applications, particularly in environments where fixed infrastructure is either unavailable or impractical. Equipped with onboard processing capabilities and sensors, such as cameras, radar, and RFID readers, UAVs can independently handle real-time data acquisition and edge computation. Amodu et al. (2023) emphasize the crucial role of UAVs integrated with mobile edge computing to enable efficient data acquisition, processing, and communication in real-time scenarios [2]. This unique capability allows UAVs to perform end-to-end data collection, processing, and communication, making them indispensable for a wide range of mission-critical applications, particularly in environments where real-time performance is essential. For instance, in precision agriculture, UAVs equipped with multispectral sensors and onboard data processing capabilities monitor vast farmland areas, providing insights on crop health to optimize resource use and increase yields [4]. In search and rescue operations, UAVs equipped with thermal imaging sensors can rapidly scan large regions, providing real-time data on survivor locations [20]. Likewise, in urban environments, UAVs can analyze real-time video feeds to detect traffic congestion, accidents, or hazards, enabling quick interventions to improve traffic flow and safety [16]. By handling both sensing and computation onboard, UAVs eliminate the need for fixed edge servers, functioning as adaptable, autonomous platforms capable of meeting the dynamic demands of modern IoT applications.

1.3 A Survey on UAV-based Routing, Placement and Task Execution Strategies

The fundamental challenges in efficiently scheduling and routing UAVs for real-time task execution are closely related to classical combinatorial optimization problems, notably the Traveling Salesman Problem (TSP), Vehicle Routing Problem (VRP), and Dial-a-Ride

Problem (DARP). To efficiently manage the spatial distribution and time-sensitive execution of UAV tasks, these optimization problems offer valuable modeling insights. One such optimization problem, the *Traveling Salesman Problem (TSP)*, seeks the shortest possible route visiting a set of locations exactly once and returning to the starting point, providing a foundational structure for single-UAV route planning [3]. On the other hand, the *Vehicle Routing Problem (VRP)* extends TSP by considering multiple vehicles originating from a depot, each serving a subset of locations, thus naturally aligning with multi-UAV deployment scenarios [24]. Further, the *Dial-a-Ride Problem (DARP)* is a dynamic version of VRP that generalizes VRP by introducing time-window constraints and pairing pickup and delivery points, making it particularly relevant for UAV-based real-time task execution where both timing and capacity constraints are critical [9].

In literature, the VRP and its variations have been widely studied, resulting in rich modeling frameworks and solution strategies that are highly relevant to UAV-based systems. For example, the *Capacitated VRP (CVRP)* imposes load capacity constraints on vehicles, while the *Capacitated Electric Vehicle Routing Problem (CEVRP)* incorporates limited driving ranges and non-linear charging behaviors, modeling the operational challenges faced by electric vehicles (EVs) [14, 15]. These VRP variants have seen extensive application in domains such as logistics, supply chain management, public transportation, and more recently, green transportation networks. Major industries including DHL, FedEx, XPO, and JD Logistics deploy VRP-based solutions for optimizing fleet operations under resource and energy constraints. Similarly, DARP has been instrumental in designing and optimizing on-demand ride services, paratransit operations, and dynamic shuttle systems, where tasks (pickups/deliveries) arrive dynamically, and operational schedules must adapt in real time. Importantly, these VRP/DARP modeling concepts extend beyond logistics or transportation and are equally applicable to mobile robotic systems like Unmanned Aerial Vehicles (UAVs), Terrestrial Mobile Robots (TMRs) and Underwater Unmanned Vehicles (UUVs).

Drawing inspiration from these classical optimization problems, this work majorly concentrates on the real-time execution of a given set of spatially dispersed, static, periodic/aperiodic control tasks using a minimum number of UAVs, by jointly optimizing per-UAV routing, task hovering durations and deadline-constrained task execution. Despite the numerous advantages, UAV-based IoT systems encounter significant challenges in reliably executing tasks within strict real-time deadlines. Efficient deployment and routing strategies are crucial for UAVs to effectively cover large areas or manage numerous spatially distributed tasks. Typically, these strategies are divided into static and dynamic approaches. Many deployments position UAVs at fixed locations within statically defined layouts, such as at the intersection points of logical grids. However, these static methods often lack flexibility to adapt to changes in user demand, task urgency, or environmental conditions [12]. In contrast, dynamic strategies allow UAVs to adjust their positions and routes in real-time to improve coverage, reduce latency, and enhance network reliability [32, 31, 29, 28].

Zhou et al. (2018) proposed an online multi-UAV repositioning methodology to dynamically adjust coverage based on emerging temporary events [32]. Building on this, Zhang and Duan (2019) introduced a fast UAV placement method to enhance wireless coverage and connectivity in areas with limited infrastructure [31]. The work by Zeng et al. (2016) optimized UAV paths to maximize data throughput in mobile relay systems [29]. To further improve communication efficiency, Zeng et al. (2018) designed UAV multicast schemes to minimize transmission time when delivering data to multiple ground users [28].

Recently researchers have also delved towards optimizing combined routing and hovering strategies for UAVs with the objective of enhancing distributed sensing and computation [2, 26, 8, 27]. Yu et al. (2020) developed an approach for using UAVs as mobile edge

computing nodes and focused on joint task offloading and resource allocation to optimize energy efficiency and computational load balancing [27]. Yang et al. (2020) and Chen et al. (2024) designed real-time routing algorithms, which also take into account necessary hovering time for the execution of aperiodic real-time tasks using UAVs [26, 8]. The objectives of these works are to maximize the number of tasks that can be completed before their deadlines, and/or minimize energy consumption.

1.4 This Work: *USRART-P, USRART-AS, USRART-AA*

In this work, we consider both periodic as well as aperiodic, real-time tasks that are spatially distributed at fixed locations across a remote isolated region. Applications such as precision agriculture, search and rescue operations, traffic management, etc. are suitable candidates for these types of tasks. Applications such as say, continuous crop health monitoring and controlled fertilizer/ water delivery (in precision agriculture), are usually persistently repeating with specific sampling periodicities. Moreover, the application set may consist of multiple instances of the same type of application; for precision agriculture say, different application instances may be employed to serve distinct agricultural fields at different locations. In any case, the applications being static and periodic, the overall system has a hyperperiodically repeating cyclic nature. Within a hyperperiod, an application executes for a certain number of task instances. For such a scenario, this work considers the employment of a set of UAVs as real-time processing/ sensing resources, and proposes an efficient heuristic called *UAV Scheduling and Routing Algorithm for Real-time Tasks - Periodic Arrivals (USRART-P).*

Further, adapting this framework for aperiodic task scenarios, we develop two additional heuristic strategies: $\boldsymbol{USRART-SA}$ for *Synchronous Aperiodic Arrivals* (common arrival time, distinct deadlines) and $\boldsymbol{USRART-AA}$ for *Asynchronous Aperiodic Arrivals* (distinct arrival times and deadlines). *USRART-SA* is applicable to scenarios where multiple aperiodic tasks are triggered simultaneously by a single critical event. A representative example of such a scenario can be *industrial facility accident management.* For instance, in the event of a fire outbreak at a chemical plant at $t = 0$, multiple UAV tasks such as toxic gas sensing, pipeline integrity checks, and perimeter thermal scans, may be simultaneously launched to assess the situation. These tasks may have distinct urgency driven deadlines. For example, toxic gas sensing can be considered the most urgent and may have a shorter deadline compared to pipeline integrity check and thermal scan. *USRART-SA* is designed to manage such synchronously triggered yet deadline-diverse tasks, ensuring timely and prioritized execution.

Certain aperiodic tasks may have known arrival times within a specified time horizon, enabling higher scheduling flexibility compared to systems with synchronous task arrivals. In such scenarios, *USRART-AA* can be usefully applied. An example scenario arises with *UAV-based last-mile delivery in smart logistics.* In this context, package pickup and drop-off requests are specified with known release times and associated delivery deadlines within the operational horizon. For instance, it may be pre-scheduled that an urgent medical supply delivery task becomes available at time $t = 5$, while a food delivery request is set to arrive at $t = 12$. Further, medical supply delivery being more urgent may have a shorter relative deadline compared to food delivery. For such an use-case, *USRART-AA* can proactively schedule UAV routes, ensuring that each request is served within its deadline while maintaining efficient utilization of the UAV fleet.

All three variations of *USRART* assume that each member of the set of UAVs starts from the same depot location at the beginning of a time horizon and flies over a subset of the task locations following a designated route, returning back to the depot by the end of that time horizon. During the traversal in its route, a UAV follows an alternating sequence of movement and hovering operations. The movement operation takes a UAV from one task

location to another. At each task location, the UAV hovers for a specific time duration during which, it performs sensing and processing for the task. With the aim of minimizing the number of UAVs required, this work proposes an efficient methodology for generating *per-UAV routes and deadline-meeting execution schedules for all tasks, aiming to maximize the subset of tasks that can be covered by each route.* **The major contributions of this work are thus summarized as follows**:

- This is possibly the first research work that presents a mechanism for UAV-based execution of a set of geographically distributed, periodic as well as aperiodic real-time applications. We have systematically analyzed the underlying problem structure and represented it in the form of a constraint optimization formulation.
- We have proposed an efficient but low-overhead heuristic approach called "**UAV Scheduling and Routing Algorithm for Real-time Tasks (*USRART*)**", for the problem at hand. With the aim of minimizing the number of UAVs, $USRART$ generates routes with deadline-meeting task schedules for each UAV.
- We propose three adaptations of $USRART$ namely, $USRART-P$ for periodic real-time tasks, $USRART-SA$ for synchronously arriving aperiodic tasks and $USRART-AA$ for asynchronous aperiodic tasks with known arrival times.
- We have conducted extensive simulations to rigorously evaluate our algorithms using diverse datasets inspired by plausible real-world scenarios. Our experiments span a wide range of conditions, varying task densities, spatial distributions, task execution times, and UAV speeds. The experimental results show that all variations of $USRART$ consistently deliver appreciable performance across diverse test cases, highlighting their robustness, generic nature, and practical applicability.

The remainder of this paper is organized as follows: Section 2 presents the system model and problem formulation. Section 3 provides a detailed description of the proposed algorithm. Finally, Section 4 discusses the experimental setup and results, demonstrating the effectiveness of our proposed approach.

2 System Model

In this section, we formulate the system models for both periodic and aperiodic real-time CPS tasks targeted for UAV-based execution. We first present the model for periodic tasks, laying the foundation for subsequent extensions to handle aperiodic task arrivals. For the first case we consider a set of n periodic real-time applications, denoted by $T = \{T_1, T_2, \ldots, T_n\}$, where each application T_i is characterized by a four-tuple $\langle loc_i, E_i, P_i, D_i \rangle$. Here, loc_i denotes the geographical location of application T_i, E_i and P_i denotes the execution requirement and period of T_i respectively. Data communication overhead between a UAV and T_i's ground location loc_i, is assumed to be incorporated within the execution time E_i of T_i. We denote by H the hyperperiod of the set of T applications; thus, $H = LCM(P_1, P_2, ...P_n)$. Given that all applications are ready to execute at the system start time "$t = 0$", H represents the smallest interval after which the arrival of all applications simultaneously synchronize again. The number of jobs of any application T_i within H is given by $\phi_i = \frac{H}{P_i}$, with the jobs being $T_{i1}, T_{i2}, \ldots, T_{i\phi_i}$. The total number of jobs to be executed within H is given by l; $l = \sum_{i=1}^{n} \phi_i$.

A fleet consisting of λ homogeneous Unmanned Aerial Vehicles (UAVs)

$$U = \{U_1, U_2, \ldots, U_\lambda\}$$

having same computational capacity and identical flight speed s, is deployed to execute the list of l jobs. At the beginning of each hyper-period, all UAVs start from a common depot location loc_D (Refer Figure 1). Then they follow distinct UAV-specific routes moving and

hovering over a subset of job locations, and finally return back to the depot by the end of the hyper-period (as depicted in Figure 1). The depot location thus act as both the origins and ultimate destinations of all UAV routes.

The UAVs are oriented towards the execution of individual jobs; that is, two distinct jobs say, T_{ij} and T_{ik} of the same application T_i, may be executed by different UAVs. Thus, we represent all jobs of all applications within the duration H, as a single sequential list of task instances $I = \langle I_1 \ldots, I_{\phi_1}, I_{(\phi_1+1)}, \ldots, I_{(\phi_1+\phi_2)}, \ldots, I_l \rangle$. The sequence I begins with jobs from application T_1, followed by T_2, and continues sequentially through T_n. The task instances, $\langle I_1, \ldots, I_{\phi_1} \rangle$ correspond to the jobs $\langle T_{11}, \ldots, T_{1\phi_1} \rangle$, instances $\langle I_{\phi_1+1}, \ldots, I_{\phi_1+\phi_2} \rangle$ represent jobs $\langle T_{21}, \ldots, T_{2\phi_2} \rangle$, and so forth, with $\langle I_{l-\phi_n+1}, \ldots, I_l \rangle$ denoting $\langle T_{n1}, \ldots, T_{n\phi_n} \rangle$. Moreover, as all UAVs start and finish their journeys at the depot, we add λ origin and ultimate destination dummy task instances $\langle I_{o1}, \ldots I_{o\lambda} \rangle$ and $\langle I_{\delta 1}, \ldots I_{\delta\lambda} \rangle$ respectively, corresponding to the λ UAV routes.

Each UAV covers a disjoint subset of task instances in its route. A UAV route $r_i = \{r_{i0}, r_{i1}, r_{i2}, \ldots, r_{in_i}, r_{in_i+1}\}$, represents an ordered list of n_i task instances assigned to UAV U_i ($\sum_{i=1}^{n} n_i = l$), where $r_{ij} = k$ indicates that the j^{th} location in U_i's route is dedicated to task instance I_k. Here, $r_{i0} = oi$ and $r_{in_i+1} = \delta i$. Route set $RS = \{r_1, r_2 \ldots r_\lambda\}$ refers to the collection of λ UAV routes. Each task instance $I_j \in I$ is described by a 10-tuple: $\langle I_j^{aid}, I_j^{tid}, I_j^{rt}, I_j^{d}, I_j^{e}, I_j^{loc}, I_j^{trt}, I_j^{st}, I_j^{ct}, I_j^{s} \rangle$ which defines its execution and scheduling related parameters. We now describe the significance of each of these ten parameters of a task instance I_j:

(i) Application id (I_j^{aid}): Identifier of the application $T_{I_j^{aid}}$ of which I_j is a task instance.

(ii) Job id (I_j^{tid}): Identifier of the job $T_{(I_j^{aid}, I_j^{tid})}$ which corresponds to the instance I_j.

(iii) Release Time (I_j^{rt}): Represents the earliest time at which instance I_j becomes available for execution. Since I_j is the $(I_j^{tid})^{th}$ job of the periodically repeating application $T_{I_j^{aid}}$, its release time is computed as:

$$I_j^{rt} = (I_j^{tid} - 1) \cdot P_{I_j^{aid}} \tag{1}$$

(iv) Deadline (I_j^{d}): Latest time by which execution of I_j must complete. It is given by:

$$I_j^{d} = I_j^{rt} + P_{I_j^{aid}} \tag{2}$$

(v) Execution Time (I_j^{e}): The execution time of all instances of a task is same. Hence, $I_j^{e} = E_{I_j^{aid}}$.

(vi) Location (I_j^{loc}): The location of I_j is derived from the location of task $T_{I_j^{aid}}$. That is, $I_j^{loc} = loc_{I_j^{aid}}$.

(vii) Travel Time (I_j^{trt}): Let, I_j be the k^{th} task instance in UAV U_α's route r_α; that is, $r_{\alpha k} = j$. Now, travel time is the time required to reach I_j^{loc} ($I_{r_{\alpha_k}}^{loc}$) from $I_{r_{\alpha(k-1)}}^{loc}$, the location of the task instance covered by the $(k-1)^{th}$ element $r_{\alpha(k-1)}$ of route r_α.

$$I_j^{trt} = D(I_{r_{\alpha(k-1)}}^{loc}, I_j^{loc})/s \tag{3}$$

In the above equation, $D(I_{r_{\alpha(k-1)}}^{loc}, I_j^{loc})$ is a function which returns the euclidean distance between the task instances covered by $r_{\alpha(k-1)}$ and $r_{\alpha k}$ and s is the speed of UAV U_α.

(viii) Start Time (I_j^{st}): The start time of I_j represents the instant when its execution begins on U_α. Now, start time I_j^{st} is given by:

$$I_j^{st} = \max\left(I_j^{rt}, I_{r_{\alpha(k-1)}}^{ct} + I_j^{trt}\right) \tag{4}$$

Here, $I^{ct}_{r_{\alpha(k-1)}}$ denotes completion time of the instance covered by $r_{\alpha(k-1)}$. Therefore, I^{st}_j is obtained as the higher between the time instant at which I_j is released and the time instant at which U_α reaches I^{loc}_j.

(ix) Completion Time (I^{ct}_j): This is represented as:

$$I^{ct}_j = I^{st}_j + I^e_j \tag{5}$$

(x) Slack Time (I^s_j): Slack denotes the maximum duration by which start of I_j can be delayed while ensuring deadline adherence. It is defined as:

$$I^s_j = I^d_j - I^{ct}_j \tag{6}$$

Finally, for any origin dummy task instance I_{oi} (corresponding to UAV U_i), $I^{aid}_{oi} = I^{tid}_{oi} = I^{rt}_{oi} = I^e_{oi} = I^{wt}_{oi} = I^{st}_{oi} = I^{ct}_{oi} = 0$, $I^{loc}_{oi} = loc_D$ and $I^d_{oi} = H$. Similarly, for any destination instance $I_{\delta i}$, $I^{aid}_{\delta i} = I^{tid}_{\delta i} = I^{rt}_{\delta i} = I^e_{\delta i} = 0$, $I^{loc}_{\delta i} = loc_D$ and $I^d_{\delta i} = H$.

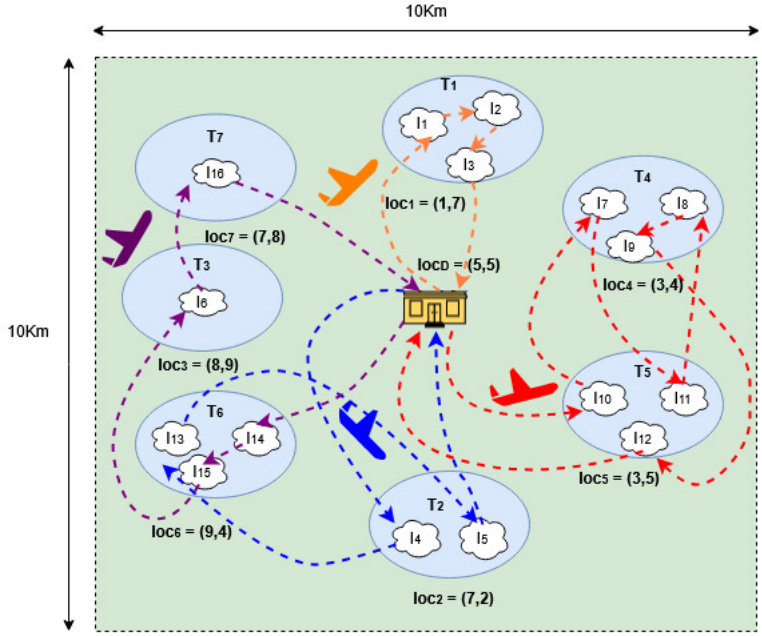


Figure 1 Multi-UAV Scheduling and Routing for Spatially Distributed Tasks.

2.1 An Example Application and System Scenario

We consider here, a real-time military surveillance system described as follows. The system requires to surveil seven geographically scattered sites ($loc_1, \ldots, loc_7$) within a given region of size 10 km × 10 km, as shown in Fig. 1. Each site is surveilled by a UAV which periodically comes and performs surveillance. This surveillance job is conducted by scanning and collecting information (such as potential enemy movements and/or rescue needs) from the area of the site, with the help of camera-equipped UAVs or via cameras installed at terrestrial locations. In addition, the UAVs may also be equipped with processing power in order to say, perform periodic adjustments of the pan-tilt-zoom settings of the installed cameras, based on received image/video feeds. The activities in a job thus include scanning, data transmission (information collection) and computation. The potential sequence of

jobs $\langle T_{i1}, T_{i2}, \dots \rangle$ associated with a particular site say loc_i, constitutes a distinct periodic real-time application T_i. The execution time E_i (of T_i) depends on the total time required by a UAV to perform all the mentioned activities associated with a job say, T_{ij}. This execution time depends on factors such as, area of the site to be scanned, number of installed cameras, size of transmitted data and computation time required to generate control outputs say, new pan-tilt-zoom setpoints. *Periodicity* of surveillance may depend among other factors on the sensitivity of the site w.r.t surveillance. Thus, high-risk sites which are say, prone to enemy activity, may require to be more frequently surveilled at shorter intervals, compared to others.

Table 1 depicts the execution times, periods, and site locations of all the seven periodic applications considered in the problem. The hyper-period of these applications is $H = 30$. The number of jobs of each application within one hyper-period has also been shown in Table 1. The objective of the problem is to deploy the minimum among a set of available UAVs which are homogeneous in terms of their flight speed (here, 30 km/h), processing capability, as well as data collection/transmission rate. All deployed UAVs should start from their designated depot location ($loc_D = (5, 5)$) at the beginning of each hyper-period, cover a subset of jobs, and return back to the depot by the end of the hyper-period. Fig.1 shows a solution to this problem scenario using the proposed algorithm $USRART$ (described later). It may observed that $USRART$ requires the deployment of four UAVs to solve this periodic real-time routing cum scheduling problem. The above running example is motivated by the real-world military surveillance operation

The above example is motivated from the requirements of modern day military surveillance (as detailed in [18]) which underscore the critical need for efficient UAV routing and task allocation in remote, high-risk environments.

Table 1 Task Parameters and Instances in the Running Example.

Application	Location	Period (mins)	Exec. Time (mins)	No. of Jobs
T_1	$loc_1 = (1, 7)$	$P_1 = 10$	$E_1 = 1.0$	$\phi_1 = 3$
T_2	$loc_2 = (7, 2)$	$P_2 = 15$	$E_2 = 0.5$	$\phi_2 = 2$
T_3	$loc_3 = (8, 9)$	$P_3 = 30$	$E_3 = 2.0$	$\phi_3 = 1$
T_4	$loc_4 = (3, 4)$	$P_4 = 10$	$E_4 = 1.5$	$\phi_4 = 3$
T_5	$loc_5 = (3, 5)$	$P_5 = 10$	$E_5 = 1.0$	$\phi_5 = 3$
T_6	$loc_6 = (9, 4)$	$P_6 = 30$	$E_6 = 1.3$	$\phi_6 = 1$
T_7	$loc_7 = (7, 8)$	$P_7 = 10$	$E_7 = 1.9$	$\phi_7 = 3$

System Model Adaptations for Handling Aperiodic Real-Time Applications

Unlike periodic tasks, aperiodic applications are modeled as single-instance tasks. The system model considers l aperiodic tasks $I_1 \dots I_l$, each arriving at arbitrary times rather than following a periodic pattern. The attributes associated with an aperiodic task has however been kept exactly same as that of a single periodic task instance, discussed above; an aperiodic task I_j is described as I_j: $\langle I_j^{aid}, I_j^{rt}, I_j^{d}, I_j^{e}, I_j^{loc}, I_j^{trt}, I_j^{st}, I_j^{ct}, I_j^{s} \rangle$. The only difference is that, unlike the periodic model where each job instance carried a distinct job identifier (I_j^{tid}) corresponding to its instance in the periodic sequence, the aperiodic model omits this job ID, as each application represents a unique, single-occurrence task. The aperiodic model supports two subclasses of aperiodic arrivals in this work: $USRART - SA$ and $USRART - AA$ as described in the Introduction section. All other assumptions and structural elements of the system model – such as UAV capabilities, depot behavior, and routing formulation – remain consistent with those described for the periodic case.

An Example Application for Aperiodic Scenario

We now consider an aperiodic system where UAVs perform last-mile deliveries within a smart logistics network spanning a 10×10 km zone, identical to the setup in the periodic scenario. Each site A_j represents a one-time aperiodic delivery request with its own release time, execution time, and deadline. For instance, if all delivery requests are known at the start ($t = 0$) but have varying deadlines – such as multiple urgent packages to be dispatched in one planning window – this maps to the *USRART-SA* scenario. Alternatively, if requests arrive over time, such as a medical supply delivery becoming available at $t = 5$ and a food delivery request at $t = 12$, this aligns with the *USRART-AA* case. UAVs handle package pickup, transport, and drop-off, starting and ending at the central depot (5, 5), similar to the periodic model. This type of UAV scheduling ensures that each delivery request meets its individual timing constraints while ensuring efficient utilization of the fleet.

3 The Problem Formulation

We now describe the problem formulation that determines per-UAV routes for efficiently executing a given set of geographically distributed real-time applications, using a minimum number of UAVs. It uses a set of binary decision variables of the form $x_{ij\alpha}$ ($i \in [0, l]$, $j \in [1, l+1]$ and $\alpha \in [1, K]$).

Among them, the subset of decision variables having type $x_{0j\alpha}$ involves the origin task instance for a UAV U_α. Similarly, the decision variables $x_{i(l+1)\alpha}$ are those which involve the dummy task instance at the final destination of a UAV U_α. Any variable $x_{ij\alpha} = 1$, denotes that the α^{th} UAV directly moves to task instance I_j from I_i; $x_{ij\alpha} = 0$, otherwise. Next, we discuss the objective function and constraints.

Objective Function. Given an upper bound K on the number of available UAVs, the objective is to determine the minimum number λ of UAVs, which are necessary to execute a geographically distributed set of real-time tasks. Let $U_1, ..., U_\lambda$ form the set of active UAVs while $U_{\lambda+1}, ..., U_K$ form the set of inactive UAVs that are not actually deployed. Additionally let, $a = \sum_{i=0}^{l} \sum_{j=1}^{l+1} x_{ij\alpha}$, for the α^{th} UAV. Now, it may be observed that if the α^{th} UAV has non-zero route length (active), then $a \geq 1$ and, $\lfloor 1/\lfloor e^a \rfloor \rfloor = 0$. Hence, the objective function becomes:

$$\textbf{Minimize } \boldsymbol{\lambda}, (1 \leq \lambda \leq K) \mid \sum_{\alpha=1}^{\lambda} \lfloor 1/\lfloor e^a \rfloor \rfloor = 0 \tag{7}$$

Constraints. There are four constraints in this formulation.

1. **Unique UAV, unique source constraint:** For any instance I_j, there is a unique UAV U_α which arrives to it after immediately serving a unique previous instance I_i.

$$\sum_{\alpha=1}^{K} \sum_{i=0, i\neq j}^{l} x_{ij\alpha} = 1, \quad \forall j \in [1, l+1] \tag{8}$$

2. **Unique UAV, unique destination constraint:** For any instance I_i there is a unique UAV U_α which serves it and then moves to a unique destination instance I_j.

$$\sum_{\alpha=1}^{K} \sum_{j=1, j\neq i}^{l+1} x_{ij\alpha} = 1, \quad \forall i \in [0, l] \tag{9}$$

3. **Non-zero route length constraint:** To ensure the generation of non-zero-length routes for all active UAVs, we enforce the following,

$$\forall \alpha, \quad x_{0(l+1)\alpha} = 0 \tag{10}$$

4. **Deadline constraint:** Each task instance I_i should complete execution before its respective deadline. To mathematically represent this constraint, we use the route elements $r_{\alpha j}$ ($1 \leq \alpha \leq K$, $0 \leq j \leq n_\alpha + 1$; discussed above) as auxiliary variables. It may be observed that all $r_{\alpha j}$ values can be derived for a fixed valuation of the decision variables $x_{ij\alpha}$. Additionally, we set: $\forall \alpha$, $r_{\alpha 0} = o\alpha$. Thus, the 0^{th} route element of a UAV U_α corresponds to its origin task instance $I_{o\alpha}$. Similarly, we set: $\forall \alpha$, $r_{\alpha(n_\alpha+1)} = \delta\alpha$, indicating that the last route element of U_α is $I_{\delta\alpha}$. Given the values of $r_{\alpha j}$, the deadline constraint can be written as:

$$I^{ct}_{r_{\alpha j}} \leq I^{d}_{r_{\alpha j}}, \quad \forall \alpha \in [1, K], \forall j \in [1, n_\alpha + 1] \tag{11}$$

Equations 2 and 5 above discuss the mechanisms for determining values of $I^d_{r_{\alpha j}}$ and $I^{ct}_{r_{\alpha j}}$. It may be observed that by enforcing constraint 11 for $j = n_\alpha + 1$, for any UAV U_α, we restrict the total route time for U_α to be upper bounded by H.

The above problem formulation involves a large number of variables and constraints making it computationally very expensive. Hence, we have not used standard optimizers (like CPLEX [13]) to solve the problem. We highlight that the main motivation to formally represent the problem as above is to clearly show the nature/structure of the scheduling problem being solved.

4 Proposed Algorithm: *USRART*

The *UAV Scheduling and Routing Algorithm for Real-time Tasks* (*USRART*; Algorithm 1) is a heuristic algorithm aimed at efficiently allocating a minimum number of UAVs for executing all task instances of a given set of independent periodic/ aperiodic applications. We present three tailored variants of the *USRART* framework, namely *USRART-P*, *USRART-SA*, and *USRART-AA*, each designed to address specific scheduling scenarios as detailed below.

$USRART - P$ first generates a single list of all task instances (of all these applications) that are spawned within one hyper-period duration. For all task instances, attributes such as release time, location, execution time, and deadline are noted. Initially, these task instances are unmapped, that is, not allocated to any UAV for their execution. Next, $USRART - P$ iteratively calls a function called *UAV Route Generator* (*URG*; Algorithm 2) until all task instances have been mapped to its designated UAV.

The function *URG* constructs the route for a single UAV. This route starts from the depot at the beginning of the hyper-period and ultimately returns back to the depot at the end of the hyper-period. As discussed earlier, a route is a sequence of task instances which are allocated for execution to this UAV. The time consumed by a generated route (t_c) consists of the UAV's total travel time, as well as its aggregate waiting times and execution times for all task instances in the route. Obviously, the value of this route time t_c is upper bounded by H, the hyper-period duration. The route is constructed by iteratively adding additional task instances in a sequential manner, until no more task instances can be added while not violating the route time constraint. At any instant after allocating a task at a certain location during partial route generation, *URG* attempts to find the best unmapped task instance (having highest *Goodness*; refer equation 12) that should be appended to the current partial route. We now provide a more detailed step-by-step explanation of both the algorithms, namely $USRART - P$ and *URG*.

4.1 Function $USRART\text{-}P$: Detailed Description

$USRART$ takes as input a set $T = \{T_1, T_2, \ldots, T_n\}$ of n applications. The output of the algorithm is a route set (RS), which contains the routes for all deployed UAVs. The algorithm begins with four initialization steps. **Line 1** initializes the global list of task instances (I) from the input application set (T) and computes essential attributes for each task instance $I_j (\in I)$ including, *release time* (I_j^{rt}), *deadline* (I_j^d), *execution time* (I_j^e), and *location* (I_j^{loc}) (Refers Section II, *System Model*). In **Line 3** we initialize a boolean array map[] of size $l = |I|$. An element map$[j] = 1$, indicates that task instance I_j has already been mapped to a UAV route; map$[j] = 0$, if I_j is still unmapped. **Line 4** initializes a global array thread[], which threads together the first instances of each application T_i within list I. **Lines 5-8** initiate an iterative process where UAVs are deployed sequentially until all task instances are assigned. In each iteration, the URG (UAV Route Generator) function is called to compute an efficient route r_α based on task constraints, for a new UAV U_α. The route is then added to the UAV route set RS, and the process repeats until all tasks in I are mapped. *Example Continued:* Following table 1, we generate a set of task instances $I = \{I_1, I_2 \ldots I_{16}\}$, derived from the given set of seven applications. For each I_j, the algorithm computes I_j^{rt}, I_j^d, I_j^e and I_j^{loc}. As example, for $I_5(I_{(\phi_{1+2})}$; 2^{nd} instance T_{22} of T_2) we have, $I_j^{rt} = 30$, $I_j^d = 45$, $I_j^e = 0.5$ and $I_j^{loc} = (7,2)$. The array `thread[]` which holds position indices (within list I) of the first instances of each application, gets initialized as `thread[]` $= \langle 1, 4, 6, 7, 10, 13, 14\rangle$.

Algorithm 1 USRART-P (T).

Input: Application Set T
Output: Set RS of UAV Routes
1 Initialize: Task Instance List I from T and compute $I_j^{rt}, I_j^d, I_j^e, I_j^{loc}$ for all $I_j \in I$
2 Initialize: $\alpha \leftarrow 0, RS \leftarrow \{\}$; `// α: current UAV count; RS: set of UAV routes`
3 Initialize: array map[] $\leftarrow [0, 0, \ldots, 0]$ of size l;
`// l = |I|; map[j] = 1, if` I_j `is mapped to a UAV`
4 Initialize: thread[] $\leftarrow [1, (\phi_1 + 1), \ldots, (l - \phi_n + 1)]$;
`// Threads together the first instances of each application` T_i `within list` I
5 **while** *there are unmapped instances in map*[] **do**
6 Increment α; `// Add a new UAV` U_α
7 $r_\alpha \leftarrow$ URG$(\alpha,$ map[]); `// Generate route` r_α `for UAV` U_α
8 $RS \leftarrow RS \cup r_\alpha$; `// Add` r_α `to route set`
9 **return** RS;

4.2 Function URG: Detailed Description

The ***URG* (UAV Route Generator)** function is designed to generate an efficient route for each UAV (U_α). The algorithm takes two inputs: the UAV identifier α and an array map[] that tracks task instances that have not yet been assigned to any UAV. The output is a route r_α which specifies the sequence of task instances assigned to U_α.

The while loop in **Lines 5 to 28** sequentially adds new task instances to UAV U_α's route r_α as long as the total route time t_c remains less than the hyper-period H. At any time during partial route generation, the for loop in **Lines 7 to 22** attempts to append the best task instance (whose index in I is stored in $best_{id}$) among all unmapped instances over all applications, to the current partial route $r_\alpha = \langle r_{\alpha 1}, r_{\alpha 2}, \ldots, r_{\alpha k}\rangle$. For any given application T_i, the while loop in **Lines 8 to 22** first attempts to find the earliest instance of T_i that can be feasibly accommodated into U_α's route. For an unmapped candidate task instance

Algorithm 2 Function URG (α,map[]).

Input: α, map[];
Output: Route r_α for UAV U_α;
1 Initialize: $r_{\alpha 0} = o_\alpha$; // o_α is the dummy task instance at the origin of route r_α.
2 Initialize: $k \leftarrow 0$; // k denotes the current length of U_α's route r_α;
3 Initialize: $t_c \leftarrow 0$; // Current time;
4 temp_thread[] $\leftarrow$ thread[]; // Create a local copy of thread array
5 **while** $t_c < H$ **do**
6 Initialize: $best_{id} \leftarrow$ null; $best_G \leftarrow 0$;
7 **for** $i \leftarrow 1$ ***to*** n **do**
8 **while** $temp\_thread[i] \leq thread[i] + \phi_i - 1$ **do**
9 $j \leftarrow temp\_thread[i]$;
10 **if** $map[j] = 1$ **then**
11 $temp\_thread[i] \leftarrow temp\_thread[i] + 1$;
12 *continue*; // Skip current instance of T_i, move to next instance
13 **if** $I_j^s < 0$ // Refer eq.6 for slack time I_j^s
14 **then**
15 $temp\_thread[i] \leftarrow temp\_thread[i] + 1$;
16 *continue*; // Slack is negative, move to next instance
17 **if** $I_j^{ct} + D(I_j^{loc}, I_{\delta_\alpha}^{loc})/s > H$ // Refer eq.5 for completion time I_j^{ct}
18 **then**
19 $temp\_thread[i] \leftarrow thread[i] + \phi_i$;
20 *break*; // Route time overshoots, move to next application
21 **if** $G_j > best_G$ // Refer eq.12 for *Goodness* G_j
22 **then**
23 $best_{id} \leftarrow j$; $best_G \leftarrow G_j$;
24 **if** $best_{id} == null$ **then**
25 $n_\alpha = k$; $r_{\alpha n_\alpha + 1} = \delta_\alpha$; // Cannot add more instances. δ_α:destination dummy
26 Update $I_{\delta_\alpha}^{ct} \leftarrow t_c + D(I_{r_{\alpha k}}^{loc}, I_{\delta_\alpha}^{loc})/s$; *break*;
27 $r_{\alpha(k+1)} \leftarrow best_{id}$; // Add $I_{r_{\alpha(k+1)}}$ to route r_α
28 Update $t_c \leftarrow I_{r_{\alpha(k+1)}}^{ct}$; // Update current time
29 Increment $k \leftarrow k + 1$; // Update current route length
30 Mark map[$best_{id}$] $\leftarrow 1$; // Mark $I_{r_{\alpha(k+1)}}$ as mapped
31 **return** r_α;

say I_j to be eligible for inclusion into r_α at position $r_{\alpha(k+1)}$, the following conditions must be satisfied. First, the UAV which is at location $I_{r_{\alpha k}}^{loc}$ at time t_c must be able to travel to I_j's position, then either start immediately or wait (Refer eq. 4) until the release of I_j at time I_j^{rt} (Refer eq. 1), and finally be able to complete execution before I_j^d (Refer eq. 2), the deadline of I_j. The deadline check is performed as part of the condition in **Line 13**. The condition in **Line 17** is necessary to ensure that the length of route r_α remains within the hyper-period duration H after the inclusion of I_j. If a candidate instance I_j is deemed to be feasible, its *Goodness* G_j is computed in **Line 21**. The value of *Goodness* which is used to determine the most eligible candidate task instance I_j that should be added to r_α at $r_{\alpha(k+1)}$, is computed as:

$$G_j = [w_1 \cdot S_j + w_2 \cdot Tot_j + w_3 \cdot \mathcal{D}_j]^{-1} \tag{12}$$

In the above equation, S_j denotes the tentative relative slack that will remain after completion of execution, if I_j is actually selected for inclusion at $r_{\alpha(k+1)}$. The value of S_j is computed as: $S_j = I_j^s / \max_{\forall i \in T}(P_i - E_i)$, where I_j^s is the actual tentative slack (Refer

eq. 6) associated with I_j's inclusion, and $\max_{\forall i \in T}\left(P_i - E_i\right)$ denotes the maximum possible slack that may possibly be enjoyed by any task in the system. A lower value of S_j indicates a relatively higher urgency w.r.t inclusion of I_j in r_α; thus in equation 12, $G_j \propto 1/S_j$.

The value of Tot_j is computed as: $Tot_j = (I_j^{st} - I_{r_{\alpha k}}^{ct})/(\max_{\forall i,j \in T}(D(loc_i, loc_j)/s + P_j))$, where $(I_j^{st} - I_{r_{\alpha k}}^{ct})$ represents the total travel time and waiting time that must be spent by UAV U_α, after completion of the last task instance $(I_{r_{\alpha k}}^{ct})$ in its current partial route, and before the start (I_j^{st}) of I_j's execution. On the other hand, $\max_{\forall i,j \in T}(D(loc_i, loc_j)/s + P_j)$ represents the maximum travel and waiting time that can possibly be spent by a UAV U_α between the completion of execution of a task instance and start of execution of the next instance, in any feasible route r_α which does not violate deadlines. It may be noted that for a candidate instance I_j, if Tot_j is relatively higher, it indicates that the UAV wastes relatively more time travelling and waiting than more useful task execution, ultimately resulting in lower resource utilization. Thus, $G_j \propto 1/Tot_j$.

The parameter $\mathcal{D}_j$ quantitatively represents the relative density of tasks within a stipulated radius[1] γ around I_j's location. Density $\mathcal{D}_j$ is formulated as: $\mathcal{D}_j = (T^\gamma_{\max} - T^\gamma_j)/(T^\gamma_{\max} - T^\gamma_{\min})$ where, T^γ_j denotes the number of tasks within radius γ, around I_{loc_j}, while $T^\gamma_{\max}$ $(T^\gamma_{\min})$ is the maximum (minimum) number of tasks found within radius γ, across all task locations within the region being considered in the problem. For a candidate instance I_j a relatively higher value of $\mathcal{D}_j$ indicates a lower density of tasks around I_j^{loc}. It may be noted that, if a candidate I_j located in a sparsely dense area is selected for inclusion in U_α's route, the likelyhood of higher travel and waiting times in the future route of r_α, will increase. Thus, $G_j \propto 1/\mathcal{D}_j$.

In equation 12, G_j has been designed to be inversely proportional to the linear combination of S_j, Tot_j and $\mathcal{D}_j$. Here, w_1, w_2, w_3 are positive quantities; $w_1 + w_2 + w_3 = 1$. If G_j is better than the best *Goodness* seen so far ($best_G$; **Line 21**), $best_{id}$ and $best_G$ are updated (**Line 23**). After all instances have been considered, the instance $I_{best_{id}}$ is added to r_α (**Line 27**), current time and route length are updated (**Lines 28 and 29**) and $I_{best_{id}}$ is marked as mapped (**Line 30**). Finally we consider the case where no new task instances can be feasibly added to the current route $r_\alpha = \{r_{\alpha 1}, \ldots, r_{\alpha k}\}$. In this case, $best_{id}$ remains null after evaluating all task instances. We set the route length and add the destination dummy instance δ_α to route r_α, in **Line 25** and update total route time $(I_{\delta_\alpha}^{ct})$ in **Line 26**.

Example Continued: Continuing further with our running example (Refer "An Example System", Section 2, and "Example Continued", Section 4.1), Table 2 presents the final UAV routes and execution schedules generated by $USRART-P$ for the scenario described earlier. It may be observed that the solution requires four UAVs $\{U_1, \ldots, U_4\}$. Out of the 16 task instances in I, U_1 covers six instances, U_2 and U_3 covers three instances each, while U_4 covers four instances. The table also depicts the relative slack S_j, relative total time Tot_j, relative density D_j, *Goodness* G_j, start time I_j^{st} and completion time I_j^{ct} for each instance I_j. Gantt charts of the final routes and schedules have also been depicted in Fig. 2. We see that as is necessary, total route times for all UAVs is less that $H = 30$. A diagramatic representation of the final routes has been shown in Fig. 1.

4.3 *USRART-AA* and *USRART-SA*

The aperiodic variant $USRART-AA$ retains the overall structure of $USRART-P$ but introduce key changes to handle possibly distinct but known arrivals as well as deadlines. Feasibility is now checked by ensuring that each task starts no earlier than its arrival time and

[1] λ: The value of γ is an input constant to the problem and remains same for all tasks

Table 2 Generated UAV routes for the running example with associated parameter values.

UAV (U_α)	Instance I_j (T_{xy})	Rel. Slack Time (S_j)	Rel. Total Time (Tot_j)	Relative Density (D_j)	Goodness (G_j)	Start Time (I_j^{st})	Completion Time (I_j^{ct})
U_1	I_{10} (T_{51})	0.1742	0.0849	0.0	0.0947	4.0	5.0
	I_7 (T_{41})	0.0522	0.0424	0.5	0.1369	7.0	8.5
	I_{11} (T_{52})	0.2961	0.0424	0.0	0.1100	10.5	11.5
	I_8 (T_{42})	0.1742	0.0424	0.5	0.1735	13.5	15.0
	I_9 (T_{43})	0.2961	0.1061	1.5	0.4419	15.0	21.5
	I_{12} (T_{53})	0.1916	0.0424	2.0	0.4787	23.5	24.5
U_2	I_4 (T_{21})	0.2539	0.1531	0.5	0.2527	7.21	7.71
	I_{13} (T_{61})	0.5342	0.1201	1.0	0.4203	13.37	14.67
	I_5 (T_{22})	0.3196	0.1201	1.5	0.4559	20.32	20.82
U_3	I_1 (T_{11})	0.0019	0.1899	1.0	0.2955	8.94	9.94
	I_2 (T_{12})	0.3135	0.0011	2.0	0.4946	9.94	11.0
	I_3 (T_{13})	0.3135	0.1911	2.0	0.5896	11.0	21.0
U_4	I_{14} (T_{71})	0.0309	0.1531	1.5	0.3858	7.21	9.11
	I_{15} (T_{72})	0.2822	0.0188	1.5	0.3941	9.11	11.9
	I_6 (T_{31})	0.4624	0.0600	1.5	0.4687	14.72	16.72
	I_{16} (T_{73})	0.2822	0.0694	2.0	0.5194	19.56	21.9

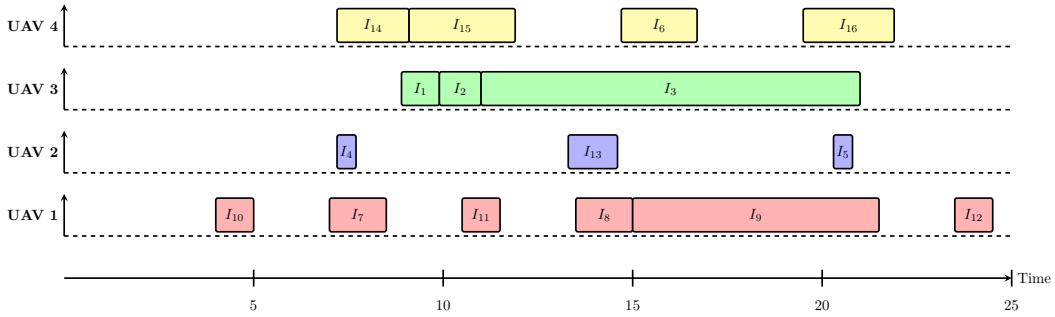


Figure 2 UAV Allocation Gantt Chart for running example.

completes before its deadline. The pseudocode for this scenario is presented in Algorithm 3. A new array, $\texttt{skip}_\alpha[]$, tracks tasks that UAV U_α cannot feasibly serve, enabling early pruning and improving efficiency. The while-loop is updated: unlike the periodic case where tasks were added while route time t_c stayed below hyper-period H, here it continues until all tasks are scheduled or skipped. Also, unlike $USRART-P$ which used a `thread[]` array to manage multiple task instances, aperiodic tasks are treated as single applications without instances. Further, in cost calculations, the parameter period in relative slack and relative total time terms (S_j, Tot_j) is replaced by the task's deadline to align with aperiodic constraints. The rest, including density formulations and overall cost structure, remains as in $USRART-P$. $USRART-SA$ is identical to $USRART-AA$ except the fact that $USRART-SA$ assumes a common arrival time unlike $USRART-AA$ which handles individual known arrivals.

5 *USRART*: Time Complexity Analysis

For the periodic task scenarios the core computational effort in ***USRART*** lies in the repeated calls to the ***URG*** function. Each invocation of URG generates a route for a single UAV. During this process, the algorithm iterates over n applications, evaluating approximately l/n task instances per application. Since computing the goodness metric for each task instance requires $O(n)$ time, the total complexity per invocation of URG is $O(n \cdot l)$. Given that URG is called up to n times within $USRART$, the overall worst-case time complexity of

Algorithm 3 USRART-SA and USRART-AA.

Input: Application Set A
Output: Set RS of UAV Routes
Initialize: $\alpha \leftarrow 0, RS \leftarrow \{\}, mapped \leftarrow 0$ // current UAV count, set of UAV routes; counter for the allocated applications
Initialize: array map[] $\leftarrow [0, 0, \ldots, 0]$ of size l // $l = |A|$; 1 means mapped, 0 means unmapped
while *there are unmapped applications in map*[] **do**
  Increment α; Initialize $r_\alpha \leftarrow \{\}, r_{\alpha 0} = o_\alpha$ // r_α is the route for UAV U_α, o_α is r_α's starting dummy application
  Initialize $\text{skip}_\alpha[] \leftarrow [0, \ldots, 0]$ // Array marking applications skipped by UAV U_α
  Initialize $\text{skipped}_\alpha, k, t_c \leftarrow 0$ // Count of skipped applications, route length, current time
  while $mapped + skipped_\alpha < l$ **do**
    Initialize $\text{best}_{id} \leftarrow \text{null}, \text{best}_C \leftarrow 0$;
    foreach *Application* $A_j \in A$ **do**
      if $map[j] == 1$ ***or*** $skip_\alpha[j] == 1$ **then**
        continue;// Skip assigned or infeasible applications
      Compute A_j^{ct};// Refer Eq. 5
      if $A_j^s < 0$// Refer Eq. 6
       then
        Mark $\text{skip}_\alpha[j] \leftarrow 1$, Increment skipped_α, **continue**;
      if $C_j > best_C$// Refer Eq. x
       then
        $\text{best}_{id} \leftarrow j, \text{best}_C \leftarrow C_j$;
    if $best_{id} == null$ **then**
      $n_\alpha = k$; $r_{\alpha n_\alpha + 1} = \delta_\alpha$; // Cannot add more applications. δ_α:destination dummy
      Update $A_{\delta_\alpha}^{ct} \leftarrow t_c + D(A_{r_{\alpha k}}^{loc}, A_{\delta_\alpha}^{loc})/s$; *break*;
    $r_{\alpha(k+1)} \leftarrow best_{id}$; Update $t_c \leftarrow A_{r_{\alpha(k+1)}}^{ct}$; Increment $k \leftarrow k + 1$, mapped $\leftarrow$ mapped $+ 1$; Mark map[$best_{id}$] $\leftarrow 1$;// $A_{r_{\alpha(k+1)}}$ mapped
  $RS \leftarrow RS \cup \{r_\alpha\}$;// Add r_α to route set
return RS

USRART is: $O(n \cdot (n \cdot l)) = O(n^2 \cdot l)$. As, $l \leq n \cdot (H/P_{\min})$, where $P_{\min} = \sum_1^n P_i$, we can express the overall complexity $O(n^2 \cdot l)$ as $O(n^3)$. For the aperiodic task scenarios, the overall time complexity remains the same, i.e., $O(n^3)$, since the algorithmic structure of $USRART$ remains unchanged.

6 Experimental Evaluation

The performance of the proposed algorithms namely $USRART - P$, $USRART - SA$ and $USRART - AA$ has been evaluated using extensive simulation based experiments. However our approach could not be benchmarked against any existing algorithm because we could not find any notable existing framework having a comparable problem model as considered in this paper. This is possibly because UAV-based remote execution of real-time CPS tasks is a very futuristic and emerging research topic. Additionally as discussed in section 3 (The Problem Formulation), design of routing and scheduling strategies for this setup is significantly complex. In the absence of a comparable state-of-the-art work, we have compared $USRART - P$ against two baseline heuristics $USRART_{B1}$ and $USRART_{B2}$ as described next:

- $USRART_{B1}$: In this baseline version, instead of using the *Goodness* metric (equation 12) to choose the best unmapped task instance that should be appended to the current partial route (Refer Algorithm 2 (Function URG)), we randomly choose a candidate task instance from the list of eligible instances.
- $USRART_{B2}$: In this case, instead of using the *Goodness* metric G_j (equation 12) used in $USRART$, here we employ a simpler *Goodness* function G'_j which disregards relative density D_j:

$$G'_j = [w_1 \cdot S_j + w_2 \cdot Tot_j]^{-1} \tag{13}$$

Next, we discuss the data generation framework for all variations of $USRART$ in detail. For the aperiodic case, we adopt only $USRART_{B2}$ as the baseline, as $USRART_{B1}$, which relies on random selection, has been observed to perform poorly across all cases in the periodic scenario. Thus, $USRART_{B1}$ is not considered for the aperiodic case.

6.1 Data Generation Framework

An exhaustive set of experiments has been performed using carefully designed synthetic datasets to evaluate the performance of $USRART$ in various plausable real-world scenarios that it may encounter in practice. To derive essential real-time task parameters such as periodicities and execution times, we have conducted relevant surveys on real-world UAV application scenarios. For example in precision agriculture, UAVs are utilized for crop health monitoring, pest detection, and irrigation management, which require detailed flight paths over agricultural fields, data capture via sensors, and real-time data analysis. Each task typically takes between *5 to 15 minutes*, considering the time required for UAVs to navigate the area, capture necessary data, and transmit it for processing [25]. Similarly, traffic management applications, where UAVs monitor traffic flow, congestion, or accidents, demand high-resolution imaging and density estimation. These tasks usually require *20 to 25 minutes* per zone, depending on the area size and urban density [11]. Other significant UAV applications include search and rescue missions, where UAVs navigate disaster zones to locate victims or deliver supplies. These operations typically require around *30 minutes*, as the UAVs must cover large areas while coordinating with on-ground teams [10]. In military contexts, UAVs are used for surveillance and reconnaissance, with missions ranging from *15 to 45 minutes*, influenced by the complexity of the task and the geographical challenges involved [18]. Additionally, UAVs are increasingly used for environmental monitoring, for tasks like air quality assessment, water resource management, and deforestation analysis, with tasks typically taking *14 to 20 minutes* for comprehensive data collection and processing [6]. Based on the above studies we have attempted to construct a data generation framework which should be able to ensure that experimental analysis is applicable to real-world scenarios while being also adaptable for future benchmarking efforts. The common parameters used in the experiments discussed in Section 6.2 below, are described as follows:

- **Application locations**: Locations are randomly generated using a uniform distribution within a fixed grid size of $10km \times 10km$ for all experiments apart from experiment 3. In experiment 3, we have generated the locations using a normal distribution as discussed later in Section 6.2.
- **Application Count**: In experiments 2 and 4, we run the heuristics for 50 applications whereas for experiments 1, 3, 5 and 6 we run them for both 50 and 100 applications.
- **Application Periods**: The period of each application, in the context of the periodic case experiments, is randomly selected uniformly from the range [10 mins, 50 mins], aligned with the application scenarios discussed above.

- **Application Deadlines**: For experiments 5 and 6, application deadlines are randomly selected within 20 to 180 minutes.
- **Application Arrival Times**: For experiments 5 and 6 (aperiodic tasks), arrival times are randomly selected within [0, 135] minutes. The range starts from 0 to align with the *USRART-SA* case (where all applications arrive at $t = 0$), while 135 minutes serves as the upper bound for varied arrival patterns.
- **Application execution times**: For experiments 1 to 4 (periodic tasks), application execution times are generated from the range [0.5 mins, 2.5 mins] using a normal distribution with mean $\mu = 1.5$ mins and standard deviation $\sigma = 0.5$ mins. To evaluate the impact of task execution time variation, execution times are progressively increased along the x-axis in steps of 10% for experiments 1, 2, and 4. For aperiodic task experiments (5 and 6), execution times also follow a normal distribution with $\mu = 10$ mins and $\sigma = 3$ mins, bounded between 0.5 and 19.5 minutes, with the same 10% incremental variation applied across both *USRART-SA* and *USRART-AA*.
- **UAV speed**: In Experiments 1,4 and 5 the UAV speed is fixed at 30 km/h (0.5 km/min) to maintain consistency in travel time calculations across different task counts. In Experiments 2,3 and 6, UAV speeds of 0.5 km/min, and 1 km/min are used to analyze the effect of the variation in speed on Average UAV Count.

Performance Metrics

The following two metrics have been used to evaluate the performance of $USRART$:

- **UAV Count**: The number of UAVs required to cover all task instances using a given heuristic algorithm, is calculated for each test case.
- **Algorithm runtime**: The proposed heuristic approaches are evaluated based on the time required to compute the UAV routes and associated task schedules.

6.2 Results

To evaluate the performance of all variations of $USRART$ and compare them against baseline heuristic strategies, six experiments have been conducted. All experiments apart from experiment 4 measure the number of UAVs required to generate UAV routes and task schedules, against increase in task execution times, variations in task location density, change in UAV speeds and variation in the number of applications. Experiment 4 on the other hand, depict runtimes of the algorithms as task execution times are varied on the x-axis. The value of each result point in the plots for any of the depicted experiments represents the average over 500 test cases derived from a designated fixed set of parameter values. In order to compute *Goodness* for $USRART - P$, the parameters S_j, Tot_j, and D_j are multiplied with empirically derived weight constants $w_1 = 0.5$, $w_2 = 0.2$, and $w_3 = 0.3$, respectively. Similarly, for $USRART_{B2}$ in the periodic setting, we use weight constants as $w_1 = 0.5$ and $w_2 = 0.5$ whereas in the aperiodic cases, we assign $w_1 = 0.4$ and $w_2 = 0.6$.

6.3 Experimental Analysis for *USRART-P*

Experiment 1: UAV Count Vs. Exec. Time, for Different #Applications

We analyze here, the impact of execution time variations on UAV deployment. In the experiment, we compare two scenarios. The first scenario consists of 50 tasks while the second scenario consists of 100 tasks distributed within the same area. For both the scenarios, UAV speed is kept fixed at 0.5 km/min. The results in Figure 3 highlight that as obvious,

number of UAVs required rises linearly with both the number of applications as well as increase in task execution times. The trends thus confirm that required UAV count increases as resource demands grow, with larger task sets exhibiting sharper increase in UAV counts. From the comparative analysis, we observe that the proposed algorithm $USRART$ performs more efficiently as compared to the other two baseline heuristics. $USRART_{B1}$, which randomly selects tasks without considering *Goodness*, results in a significantly higher UAV count compared to both $USRART$ and $USRART_{B2}$. On the other hand, $USRART_{B2}$ which ignores task density when choosing the next task instance to be appended to be a route, shows poorer performance compared to $USRART$, demonstrating the necessity of incorporating all three factors (slack time, total time, and density) in task selection.

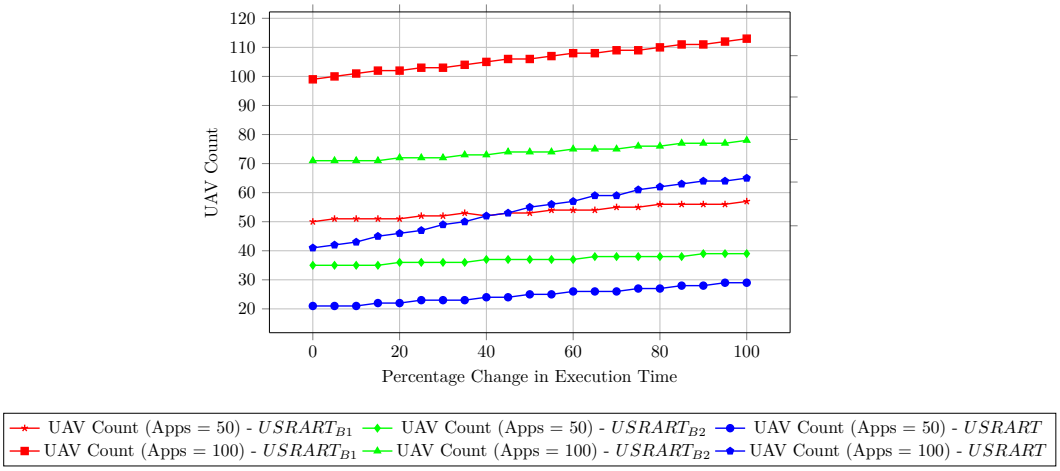


Figure 3 UAV Count Vs. Percentage Change in Exec. Time, for different #Applications.

Experiment 2: UAV Count Vs. Exec. Time, for Different UAV Speeds

This experiment evaluates the number of UAVs required to execute a set of 50 real-time periodic applications, as average execution times of these tasks is progressively increased on the x-axis. Here also we consider two scenarios, the first of which assumes speed of the UAVs to be 0.5 km/min while the second assumes the speed to be 1.0 km/min. The results in Figure 4 reveal that higher UAV speed leads to a reduction in the number of UAVs required as faster UAVs can cover more task instances within the same timeframe. However, for a fixed value of speed, a higher number of UAVs are required as average task execution times increase. Lastly similar to Experiment 1, $USRART$ performs consistently better than $USRART_{B2}$ performs better than $USRART_{B1}$.

Experiment 3: UAV Count Vs. Sparsity of Application Locations, for Different UAV Speeds

Here, we analyze the impact of the variation in application location sparsity on the number of UAVs required to cover 50 applications within a rectangular 10 km $\times$ 10 km region (left-bottom corner = (0, 0), right-bottom corner = (10, 10)). A Dataset corresponding to a certain sparsity is obtained by deriving the location values from a normal distribution having mean $(5, 5)$ and standard deviation (z, z) where z denotes a particular x-axis label ($z \in [0,5]$). For example when $z = 2$, all 50 application locations are drawn from the normal distribution having $\mu = (5, 5)$ and $\sigma = (2, 2)$; thus, the location (4, 6) can be a sample element from this

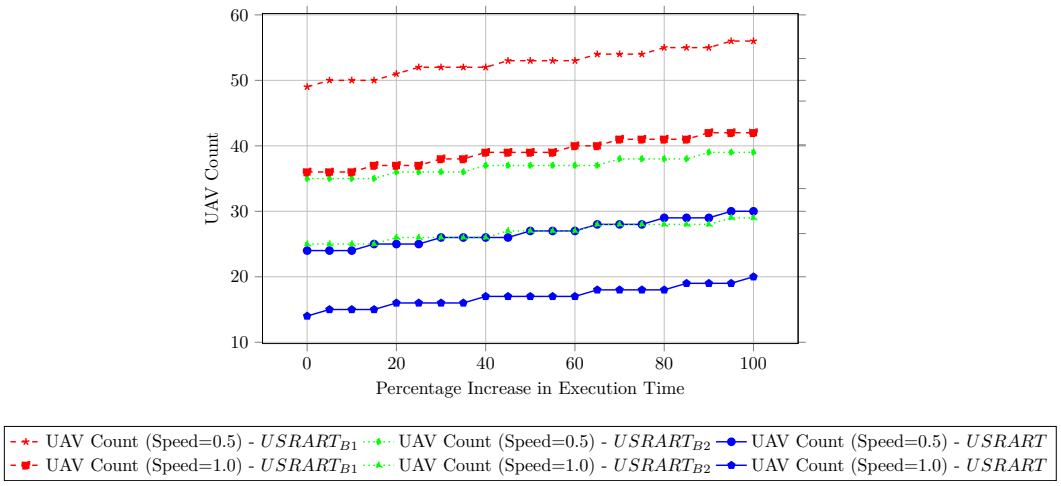


Figure 4 UAV Count Vs. Percentage Increase in Exec. Time, for different UAV Speeds.

distribution which falls within one-standard-deviation. As may be observed from the x-axis in Figure 5, we have conducted experiments with datasets having progressively higher sparsity, from $z = 0$ to $z = 5$. The special case $z = 0$ captures the scenario where all applications are concentrated only at the centroid location (5, 5). We observe that the UAV count rises as standard deviation increases. Figure 5 shows that as the standard deviation increases, the UAV count rises. A higher standard deviation spreads tasks farther from the centroid – resulting in longer travel distances and necessitating more UAVs – while a lower standard deviation concentrates tasks, reducing UAV requirements. Similar to the other experiments, we see that $USRART$ consistently outperforms both $USRART_{B1}$(random selection) and $USRART_{B2}$(ignoring density), achieving the lowest UAV count across all task sparsity levels.

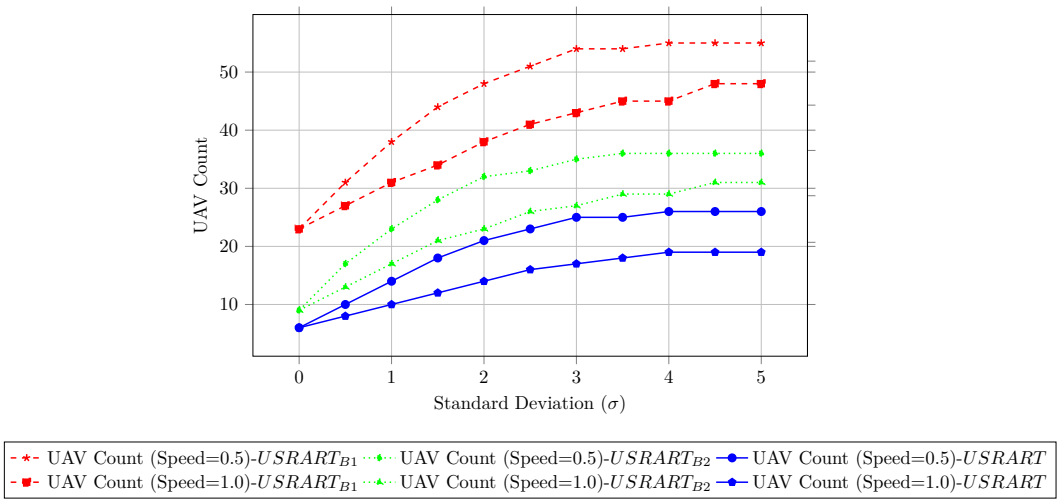


Figure 5 UAV Count Vs. Variation in Application Locations, for different UAV Speeds.

Experiment 4: Avg. Runtime vs. Exec. Time, for Different #Applications

In this experiment, we examine how variations in task execution times and application counts impact runtimes of the three heuristic algorithms evaluated. All values are measured in milliseconds. We examine two scenarios: one where 50 tasks are distributed throughout

the area, and another where 100 tasks are spread over the same region. In both cases, the UAV operates at a constant speed of 0.5 km/min. The results, illustrated in Figure 6, reveal several key observations. First, for a lower application count (50 applications), the average algorithm runtime remains relatively stable across different execution time variations. This suggests that for smaller workloads, fluctuations in application execution times do not significantly impact algorithm runtime. However, as the number of applications increases to 100, the algorithm runtime becomes noticeably higher and fluctuates more with change in the execution times of applications. A comparative analysis of the different methods highlights that due to its higher computation intensiveness $USRART$ incurs higher runtime overheads compared to both $USRART_{B1}$ and $USRART_{B2}$. This is because it uses the *Goodness* metric for the purpose of task instance selection within a partial route, which incurs higher overhead. $USRART_{B2}$, which does not account for density, shows moderate algorithm runtimes while $USRART_{B1}$ is the fastest.

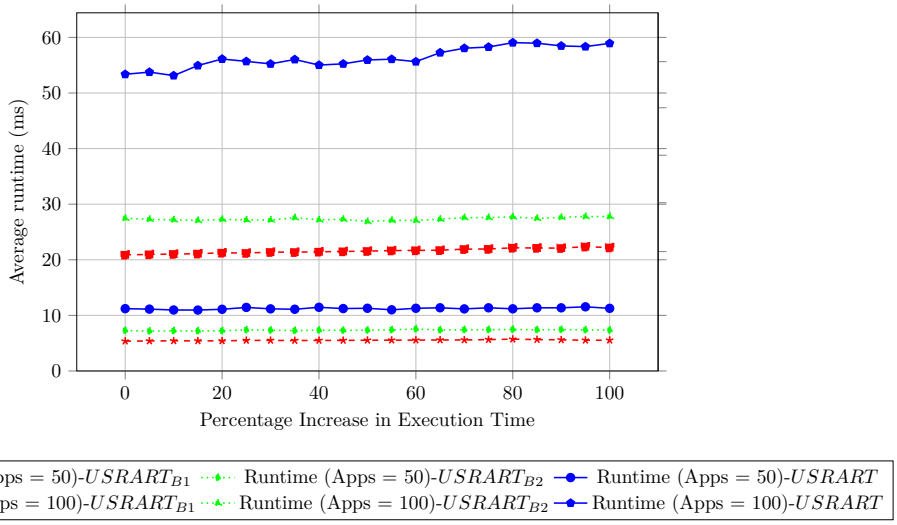


Figure 6 Effect of Average Runtime vs. Exec. Time for different number of Applications.

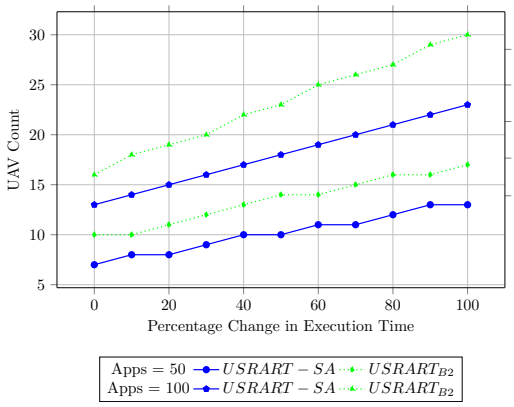
6.4 Experimental Analysis for *USRART-SA* and *USRART-AA*

Experiment 5: UAV Count Vs. Exec. Time, for Different #Applications

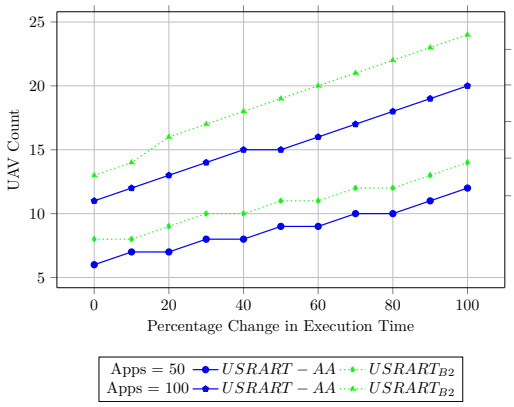
This experiment studies how increasing execution time affects UAV count, considering two scenarios (50 vs. 100 tasks) while keeping UAV speed fixed at 0.5 km/min. Figures 7a and 7b illustrate the results corresponding to $USRART$-SA and $USRART$-AA, respectively. The results show a linear increase in UAV count as execution time rises, with a sharper increase for the 100-task scenario due to higher resource demand. This trend remains consistent for both cases, demonstrating that execution time directly impacts UAV allocation, regardless of arrival time variations.

Experiment 6: UAV Count Vs. UAV Speed, for Different #Applications

This experiment evaluates how UAV speed variations affect UAV count. Figures 8a and 8b illustrate the results corresponding to $USRART$-SA and $USRART$-AA, respectively. The results confirm that reducing UAV speed lead to a higher UAV count, highlighting its importance in UAV allocation. The trends remain consistent for both cases, indicating that the effect of speed is independent of task arrival characteristics.



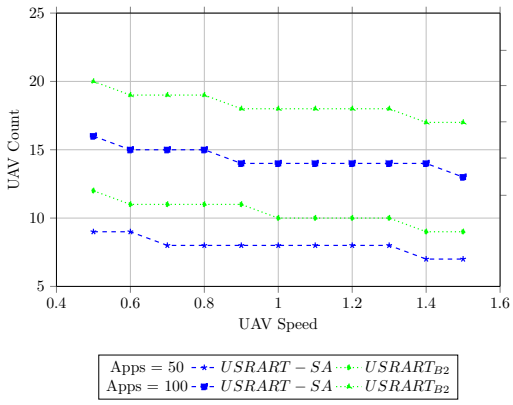
(a) Effect of Execution Time on UAV Count for Different Application Counts.



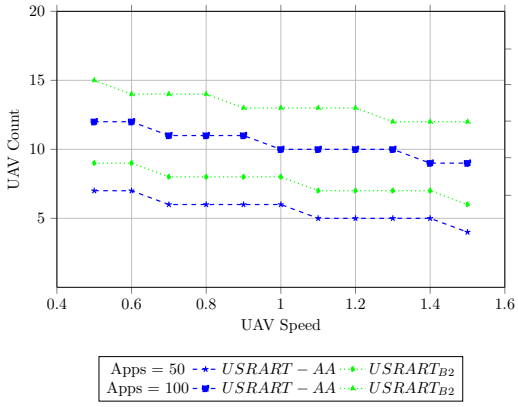
(b) Effect of Execution Time on UAV Count for Different Application Counts.

Figure 7 Effect of Exec. Time on UAV Count for different # Applications for $USRART - SA$.

Both $USRART$-SA and $USRART$-AA consistently outperform the baseline $USRART_{B2}$, achieving the lowest UAV count while maintaining efficiency. In contrast, $USRART_{B2}$, which neglects density considerations, performs noticeably worse and fails to optimize UAV distribution effectively. These results underscore the importance of jointly considering slack, total time, and density to achieve minimal UAV allocation.



(a) Effect of UAV Speed on UAV Count for Different Application Counts.



(b) Effect of UAV Speed on UAV Count for Different Application Counts.

Figure 8 Effect of UAV Speed on UAV Count for different # Applications for $USRART - AA$.

7 Discussions

To further contextualize the presented framework, we discuss two critical aspects: its generalization to broader application domains, and practical operational constraints/limitations that influence real-world UAV deployments.

7.1 Applicability of *USRART* to Other Application Domains

The methodologies developed in this work are not limited to UAV based systems but can be applied to other autonomous mobile systems operating in similar constrained and dynamic environments. Two representative domains are discussed below:

Domain 1 – Terrestrial Mobile Robots (TMRs). Terrestrial mobile robots are increasingly deployed for ground-based operations, especially in environments where human intervention is risky or inefficient [22]. A representative use-case is *hazardous area surveillance* in mapped industrial plants, post-disaster zones, or research facilities. Consider a scenario where mobile robots are dispatched from a central depot to inspect critical zones, gather environmental data (e.g., radiation, temperature), and return. These inspection tasks can be modeled either as periodic (routine surveillance at fixed intervals) or aperiodic (emergency inspections triggered by alarms). In both cases, the synchronous release models and real-time scheduling strategies proposed here can be effectively adapted to ensure timely task execution and safe robot operation.

Domain 2 – Underwater Unmanned Vehicles (UUVs). UUVs perform complex missions in harsh underwater environments where intermittent communication and energy constraints are significant challenges. A typical application is the inspection of subsea pipelines [5], cables, or waste containers. Here, UUVs are deployed from a vessel or underwater dock to a set of inspection points. Periodic inspections (scheduled dives) and aperiodic inspections (triggered by detected anomalies) mirror the task release models discussed in this work. Careful scheduling and routing are essential to accommodate energy limitations and operational deadlines, making the proposed framework directly suited to this domain.

7.2 Consideration of Practical Operational Constraints

For obtaining accurate solutions through the proposed framework, a few practical operational constraints may be needed to be considered depending on the specific deployment scenario at hand. A few important operational constraints include:

- **Battery Constraints:** UAVs have limited battery capacity, and their energy consumption varies based on flight dynamics, hovering time, and onboard computation. Accurate energy modeling is crucial to prevent mission failures due to unexpected depletion. Recent studies have proposed mathematical models to capture these energy dynamics in the context of routing problems [7].
- **Recharging Logistics:** Incorporating recharge or battery-swap stations adds further complexity to the scheduling and routing problem. This aspect aligns with the Electric Vehicle Routing Problem (EVRP), where vehicles plan their paths considering limited energy reserves and intermediate recharging stops [19]. Leveraging such ideas will be essential for extended-duration UAV missions.
- **Environmental Factors:** External environmental conditions, particularly wind and weather variability, significantly impact UAV flight stability, energy usage, and latency. Experimental studies have demonstrated how wind disturbances alter UAV performance [21]. Advanced nonlinear guidance and robust control techniques have also been proposed to ensure safe operation under strong wind conditions [23]. Integrating adaptive and robust planning mechanisms is critical for reliable UAV mission execution in varying environmental settings.

8 Conclusions and Future Work

This paper addresses the problem of UAV-based routing and scheduling for executing a set of geographically distributed, real-time Cyber-Physical System (CPS) applications. Since an optimal solution to this problem proves to be computationally expensive, we have proposed an efficient but fast heuristic strategy called *UAV Scheduling and Routing Algorithm*

for Real-time Tasks (USRART). $USRART$ systematically decomposes the problem by generating per-UAV routes and schedules, iteratively mapping tasks while maintaining deadline constraints. We propose three algorithm variants: $USRART - P$ (periodic tasks), $USRART - AS$ (synchronous aperiodic), and $USRART - AA$ (asynchronous aperiodic). $USRART - P$ generates per-UAV routes within a hyperperiod to maximize task coverage under deadline constraints. $USRART - AS$ and $USRART - AA$ iteratively allocate UAVs based on task availability and urgency, ensuring real-time guarantees. Through extensive simulations, we have demonstrated that $USRART$ maintains high performance across diverse operational scenarios, varying task distributions, execution demands, and spatial layouts. The results highlight $USRART$'s adaptability and practical utility in real-world UAV-based CPS application scenarios, particularly in resource-constrained and infrastructure-limited environments.

As future work, we plan to focus on enhancing $USRART$ by incorporating dynamic task arrivals, UAV energy constraints and collaborative multi-UAV strategies to further improve robustness and real-time adaptability in practical deployments. Furthermore, extending $USRART$ to handle stochastic execution times and travel uncertainties would make it more applicable to real-world conditions where unexpected delays are inevitable. Finally, real-world deployment and experimental validation in diverse terrain conditions would help refine the algorithm's performance and reliability. Addressing these challenges will further enhance UAV-based CPS task execution, making it a scalable and resilient solution for autonomous operations in remote environments.

References

1 Mansoor Akhtar, Muhammad Raffeh, Fakhar ul Zaman, Ali Ramzan, Sohaib Aslam, and Faisal Usman. Development of congestion level based dynamic traffic management system using iot. In *2020 international conference on electrical, communication, and computer engineering (ICECCE)*, pages 1–6. IEEE, 2020.

2 Oluwatosin Ahmed Amodu, Rosdiadee Nordin, Chedia Jarray, Umar Ali Bukar, Raja Azlina Raja Mahmood, and Mohamed Othman. A survey on the design aspects and opportunities in age-aware UAV-aided data collection for sensor networks and internet of things applications. *Drones*, 7(4):260, 2023.

3 David L. Applegate, Robert E. Bixby, Václav Chvátal, and William J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2007.

4 Muhammet Fatih Aslan, Akif Durdu, Kadir Sabanci, Ewa Ropelewska, and Seyfettin Sinan Gültekin. A comprehensive survey of the recent studies with UAV for precision agriculture in open fields and greenhouses. *Applied Sciences*, 12(3):1047, 2022.

5 Martin Aubard, Sergio Quijano, Olaya Álvarez-Tuñón, László Antal, Maria Costa, and Yury Brodskiy. Mission planning and safety assessment for pipeline inspection using autonomous underwater vehicles: A framework based on behavior trees. *arXiv preprint arXiv:2402.04045*, 2024. doi:10.48550/arXiv.2402.04045.

6 Murat Bakirci. Smart city air quality management through leveraging drones for precision monitoring. *Sustainable Cities and Society*, 106:105390, 2024.

7 Cristian Cataldo-Díaz, Rodrigo Linfati, and John Willmer Escobar. Mathematical models for the electric vehicle routing problem with time windows considering different aspects of the charging process. *Annals of Operations Research*, 2023. doi:10.1007/s10479-023-05713-z.

8 Long Chen, Guangrui Liu, Xia Zhu, and Xin Li. A heuristic routing algorithm for heterogeneous UAVs in time-constrained mec systems. In *IEEE International Conference on Communications (ICC)*, 2024.

9 Jean-François Cordeau and Gilbert Laporte. Models and algorithms for the dynamic dial-a-ride problem. *Transportation Science*, 43(2):86–99, 2009.

10 Jiong Dong, Kaoru Ota, and Mianxiong Dong. UAV-based real-time survivor detection system in post-disaster search and rescue operations. *IEEE Journal on Miniaturization for Air and Space Systems*, 2(4):209–219, 2021.

11 R Ebrahim, MM Bruwer, and SJ Andersen. Small-city traffic management using unmanned aerial vehicles (UAVs). In *Proceedings of the Southern African Transport Conference*. Southern African Transport Conference 2021, 2021.

12 Boris Galkin, Jacek Kibilda, and Luiz A. DaSilva. Coverage analysis for low-altitude uav networks in urban environments. In *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, pages 1–6, 2017. doi:10.1109/GLOCOM.2017.8254658.

13 IBM. Ibm ilog cplex optimization studio. *Online Resource*, 2024.

14 Ya-Hui Jia, Yi Mei, and Mengjie Zhang. A bilevel ant colony optimization algorithm for capacitated electric vehicle routing problem. *IEEE transactions on cybernetics*, 52(10):10855–10868, 2021. doi:10.1109/TCYB.2021.3069942.

15 Merve Keskin and Bülent Çatay. Partial recharge strategies for the electric vehicle routing problem with time windows. *Transportation research part C: emerging technologies*, 65:111–127, 2016.

16 Navid Ali Khan, NZ Jhanjhi, Sarfraz Nawaz Brohi, Raja Sher Afgun Usmani, and Anand Nayyar. Smart traffic monitoring system using unmanned aerial vehicles (UAVs). *Computer Communications*, 157:434–443, 2020. doi:10.1016/J.COMCOM.2020.04.049.

17 Jay Lee, Behrad Bagheri, and Hung-An Kao. A cyber-physical systems architecture for industry 4.0-based manufacturing systems. *Manufacturing Letters*, 3:18–23, 2015.

18 M. Anwar Ma'sum, M. Kholid Arrofi, Grafika Jati, Futuhal Arifin, M. Nanda Kurniawan, Petrus Mursanto, and Wisnu Jatmiko. Simulation of intelligent unmanned aerial vehicle (UAV) for military surveillance. In *2013 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, pages 161–166, 2013.

19 Alejandro Montoya, Christelle Guéret, Jorge E. Mendoza, and Juan G. Villegas. The electric vehicle routing problem with nonlinear charging function. *Transportation Research Part B: Methodological*, 103:87–110, 2017. doi:10.1016/j.trb.2017.02.004.

20 Robin R. Murphy, Jennifer Kravitz, Scott Stover, and Rahmat A. Shoureshi. Mobile robots in mine rescue and recovery. In *Proceedings of the IEEE 2008 International Conference on Technologies for Homeland Security*, pages 294–299, 2008.

21 Pawel Olejnik, Martin Nowak, and Cezary Zieliński. Wind impact on UAVs: Experimental study using a multi-fan low-speed wind system. *arXiv preprint arXiv:2207.12345*, 2022. arXiv:2207.12345.

22 Jane Pauline Ramirez and Salua Hamaza. Multimodal locomotion: Next generation aerial–terrestrial mobile robotics. *Advanced Intelligent Systems*, 5(1):2300327, 2023.

23 Thomas Stastny and Roland Siegwart. Nonlinear guidance for fixed-wing UAVs in strong wind conditions. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4380–4387. IEEE, 2019. doi:10.1109/IROS40897.2019.8967814.

24 Paolo Toth and Daniele Vigo. The vehicle routing problem: latest advances and new challenges. *Springer*, 2007.

25 Parthasarathy Velusamy, Santhosh Rajendran, Rakesh Kumar Mahendran, Salman Naseer, Muhammad Shafiq, and Jin-Ghoo Choi. Unmanned aerial vehicles (UAV) in precision agriculture: Applications and challenges. *Energies*, 15(1):217, 2021.

26 Zhaohui Yang, Cunhua Pan, Kezhi Wang, and Mohammad Shikh-Bahaei. Energy efficient resource allocation in UAV-enabled mobile edge computing networks. In *IEEE International Conference on Communications (ICC)*, 2020.

27 Zhe Yu, Yanmin Gong, Shimin Gong, and Yuanxiong Guo. Joint task offloading and resource allocation in UAV-enabled mobile edge computing. In *IEEE International Conference on Communications (ICC)*, 2020.

28 Yong Zeng, Xiaoli Xu, and Rui Zhang. Trajectory design for completion time minimization in uav-enabled multicasting. *IEEE Transactions on Wireless Communications*, 17(4):2233–2246, 2018. doi:10.1109/TWC.2018.2790401.

29 Yong Zeng, Rui Zhang, and Teng Joon Lim. Throughput maximization for UAV-enabled mobile relaying systems. *IEEE Transactions on Communications*, 64(12):4983–4996, 2016. doi:10.1109/TCOMM.2016.2611512.

30 Chunhua Zhang and John M. Kovacs. The application of small unmanned aerial systems for precision agriculture: a review. *Precision Agriculture*, 13(6):693–712, 2012.

31 Xiao Zhang and Lingjie Duan. Fast deployment of UAV networks for optimal wireless coverage. *IEEE Transactions on Mobile Computing*, 18(3):588–601, 2019. doi:10.1109/TMC.2018.2840143.

32 Xiaohui Zhou, Jing Guo, Salman Durrani, and Halim Yanikomeroglu. Uplink coverage performance of an underlay drone cell for temporary events. In *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6, 2018. doi:10.1109/ICCW.2018.8403634.