Abstract—This paper studies the problem of scheduling urban air mobility trips when travel times are uncertain and capacity at destinations is limited. Urban air mobility, in which air transportation is used for relatively short trips within a city or region, is emerging as a possible component in future transportation networks. Destinations in urban air mobility networks, called vertiports or vertistops, typically have limited landing capacity, and, for safety, it must be guaranteed that an air vehicle will be able to land before it can be allowed to take off. We first present a tractable model of urban air mobility networks that accounts for limited landing capacity and uncertain travel times between destinations with lower and upper travel time bounds. We then establish theoretical bounds on the achievable throughput of the network. Next, we present a tractable algorithm for scheduling trips to satisfy safety constraints and arrival deadlines. The algorithm allows for dynamically updating the schedule to accommodate, e.g., new demands over time. The paper concludes with case studies that demonstrate the algorithm on two networks.

I. INTRODUCTION

There is growing interest in utilizing urban airspace for transportation of people and goods. Both commercial mobility-on-demand operators [1] and government-sponsored research institutes such as NASA [2] are exploring such urban air mobility (UAM) solutions in cities and surrounding regions. Studies such as [3]–[8] propose various approaches to allow urban air vehicles (UAVs) to travel safely and efficiently through cities. These proposed ideas cover a wide range of possibilities such as allowing UAVs to land at vertiports or vertiports installed on roofs of existing buildings or within cloverleaf exchanges on freeways. Several simulation tools [9]–[11] have also been developed.

In this paper, we study a dynamic scheduling algorithm for UAM networks that accounts for uncertainty in travel time and limited landing capacity. Trip demands, i.e., flights, must travel through designated routes and have arrival deadlines at their destinations. We then consider the problem of scheduling flight departures to ensure that all flights arrive no later than their deadlines and that there is always an available landing spot at the destination and intermediate nodes upon arrival. The main contributions are as follows: First, we present a model for UAM networks that allows for a time-varying set of trip demands, limited landing capacity at destinations, and uncertainty in travel times that is modeled nondeterministically with lower and upper travel time bounds. Because demands are time-varying, we refer to such networks as dynamic; in the case when all demands are available at once, we call the network static and refer to a static scheduling problem. Second, we present necessary conditions for the existence of a schedule for the dynamic UAM network. For the case of a static UAM network with a star-graph topology, we show that this necessary condition is also sufficient. Third, we present a computationally efficient scheduling algorithm to compute a schedule satisfying safety constraints and arrival deadlines, and we demonstrate our approach on several case studies.

This paper extends our earlier work [12], which considered the static scheduling problem for a limited class of star-like networks. In [12], we proposed evaluating the cost of a schedule as the sum of the difference between arrival and departure time for all trips, and we showed that an optimal schedule can be obtained from a mixed integer program. This approach extends to the more general scheduling problem in this paper, however, the mixed integer program quickly becomes intractable as the number of flights increases. Instead, the algorithm proposed in this paper uses a branch-and-bound heuristic that does not rely on a mixed integer formulation and therefore may only create a suboptimal schedule. However, we demonstrate through example that our algorithm is able to quickly obtain feasible schedules with reasonable cost, i.e., in under 1 second for 200 trips in two case study networks. Moreover, the algorithm computation can be continued in search of a lower cost schedule and interrupted at any point.

In the transportation scheduling literature, prior works have considered uncertain travel times or limitations on parking capacity separately. Particularly, [13] investigates how the flow of UAVs depends on the congestion level and finds through simulations similarities with ground highway traffic with high traffic density. The paper [14] proposes an approach for traffic scheduling that can dynamically schedule flows on the link based on real-time link information.

Uncertainty in network routing problems has also been studied before for ground transportation. The paper [15] provides a literature review of such problems, and examples of more recent work are presented in [16], which studies computation of minimum-cost paths through a time-varying network and considers several classes of waiting policies. In [17], a theoretical basis for optimal routing in transportation networks with highly varying traffic conditions is provided, where the goal is to maximize the probability of arriving on time at a destination given a departure time and a time budget.

Limited parking availability has been considered in truck scheduling, where the drivers are usually required by law to park and rest after a specified amount of driving time. For
example, in [18], it is assumed that truck drivers only have access to parking spots during specific time windows, but space limitations are not considered. The paper [19] considers a similar problem with time-dependent travel times but does not take the availability of parking spots into account.

Scheduling problems have also been well-studied in the real-time systems community, e.g., in [20], [21], where jobs often are assumed to arrive with a fixed periodicity and in some models have an uncertainty in their processing time. Our fundamental limits are similar in nature and compatible with those found in these works, but our results are tailored to applications in UAM networks. For example, for finite demands, we consider scheduling to achieve prescribed deadlines without excessively early departure times.

The remainder of this paper is organized as follows: In Section III we present the dynamic UAM network model. In Section III we establish necessary conditions for the existence of a feasible schedule. We also show that in certain star-like networks, the necessary condition becomes a sufficient condition. We present an efficient algorithm for obtaining feasible schedules in Section IV while some technical details about the algorithm are presented in the Appendix. We then demonstrate in Section V the proposed algorithm on two case studies with up to 200 trips. In both networks, we are able to compute a reasonable schedule within 1 second.

A. Notation

We let \( \mathbb{N} \) denote the natural numbers without zero while \( \mathbb{N}_0 \) the natural numbers with zero, and \( \mathbb{R} \) the reals while \( \mathbb{R}_+ \) the positive reals. For a finite set \( \mathcal{A} \), we let \( \mathbb{R}^\mathcal{A} \), denote the set of vectors indexed by the elements in \( \mathcal{A} \).

II. Problem Formulation

We model an urban air mobility (UAM) network with an acyclic directed graph \( \mathcal{G} = (\mathcal{V}, \mathcal{E}) \), where \( \mathcal{V} \) is the set of nodes and \( \mathcal{E} \) is the set of links for the network. Nodes are physical landing sites for the UAVs, sometimes called vertistops or vertiports. Links are corridors of airspace connecting nodes. Each node \( v \in \mathcal{V} \) has capacity \( C_v \in \mathbb{N}_0 \), that is, there are \( C_v \) parking spots at node \( v \) where each parking spot allows at most one UAV to stay at any time. We denote the vector of capacities \( C = \{C_v\}_{v \in \mathcal{V}} \).

We define \( \tau : \mathcal{E} \rightarrow \mathbb{R}_+ \) and \( \sigma : \mathcal{E} \rightarrow \mathbb{R}_+ \) so that for all \( e = (v_1, v_2) \in \mathcal{E} \), \( \tau(e) = \max\{\tau_1, \tau_2\} \) is the tail of edge \( e \) and \( \sigma(e) = \min\{\sigma_1, \sigma_2\} \) is the head of edge \( e \). Let \( S \subseteq \mathcal{V} \) (resp., \( T \subseteq \mathcal{V} \)) be the set of nodes that are not the head (resp., tail) of any edge, \( S = \{v \in \mathcal{V} \mid \sigma(e) = v \ \forall e \in \mathcal{E}\} \) and \( T = \{v \in \mathcal{V} \mid \tau(e) = v \ \forall e \in \mathcal{E}\} \). We assume \( S \cap T = \emptyset \).

A route \( R \) is a sequence of connected links. Denote the number of links in route \( R \) by \( k_R \) and enumerate the links in the route \( 1_R, 2_R, \ldots, k_R \) and the nodes in the route \( 0_R, 1_R, \ldots, k_R \). To avoid cumbersome notation, we use \( \ell_R \) to denote both a link and its head node along a route, i.e., \( \ell_R = \sigma(\ell_R) \) for all \( \ell \in \{1, \ldots, k_R\} \); the intended meaning will always be clear from context. Thus the route links and nodes are enumerated so that \( 0_R = \tau(1_R) \) is the origin node, \( k_R \) is the destination node, and \( \sigma(\ell_R) = \tau((\ell + 1)_R) \) for all \( \ell \in \{1, \ldots, k_R\} \) ensures the sequence is connected. Further, when the route \( R \) is clear from context, we drop the superscript-\( R \) notation. We denote the set of nodes that \( R \) travels through as \( V(R) \).

We assume that, due to operational reasons, the UAVs are only allowed to travel along a set of routes \( \mathcal{R} \).

Since, in reality, the travel time depends on external factors such as weather conditions, we assume that the travel time for each link is not exact, but rather bounded by a time interval. For each link \( i \in \mathcal{E} \), let \( \tau_i \) and \( \sigma_i \) with \( \tau_i \geq \sigma_i > 0 \) denote the maximum travel time and minimum travel time, respectively, for the link, and let \( \tau \in \mathbb{R}_+^\mathcal{E} \) and \( \sigma \in \mathbb{R}_+^\mathcal{E} \) be the corresponding aggregated vectors. Once a UAV has landed at any node, it is assumed to block a landing spot for a fixed ground service time \( w \in \mathbb{R}_+ \). For ease of notation, we assume the ground service time is uniform at all nodes, but this assumption is straightforward to relax.

Definition 1 (UAM Network): A UAM network \( \mathcal{N} = (\mathcal{G}, \mathcal{C}, \mathcal{R}, \mathcal{X}, \mathcal{V}, w) \), where \( \mathcal{G}, \mathcal{C}, \mathcal{R}, \mathcal{X}, \mathcal{V}, w \) are the network graph, node capacities, routes, and minimum and maximum link travel times as defined above.

To model the demand of UAV flights in a UAM network \( \mathcal{N} = (\mathcal{G}, \mathcal{C}, \mathcal{R}, \mathcal{X}, \mathcal{V}, w) \), we assume that every flight is associated to a route \( R \in \mathcal{R} \) and stops at intermediate nodes along the route. Therefore, a demand \( j \) is a pair \((R, f)\) where \( R \in \mathcal{R} \) and \( f \in \mathbb{R}_+ \) is the latest time the UAV must arrive (i.e., land) at the destination \( k_R \in \mathcal{V} \), i.e., its deadline.

A demand profile for a UAM network is a time-varying set \( \mathcal{D}(t) = \{(R_j, f_j)\}_{j \in \mathcal{J}(t)} \) where \( \mathcal{J}(t) \) is a time-varying index set. Then \( \mathcal{J}(t) \), and, hence, \( \mathcal{D}(t) \), is monotonically increasing and is assumed finite for all finite \( t \), but may become infinite in the limit. To coordinate the operation of the UAVs, a centralized scheduler aims to schedule all available demands \( \mathcal{D}(t) \) at each time \( t \). For example, new demands may become available in batches at certain times, requiring action from the scheduler, and other demands will reach their final destination, requiring no further consideration from the scheduler, although without loss of generality completed demands remain within the set \( \mathcal{D}(t) \). Let \( \mathcal{D} := \bigcup_t \mathcal{D}(t) = \lim_{t \to \infty} \mathcal{D}(t) \) denote all demands that will ever be considered by the scheduler, and likewise let \( \mathcal{J} := \bigcup_t \mathcal{J}(t) = \lim_{t \to \infty} \mathcal{J}(t) \).

The scheduler associates to each demand a journey consisting of an assigned departure time and the set of realized arrival times along edges in the route. Therefore, a journey is updated over time as the UAV arrives at intermediate notes. Formally, for each \( j \in \mathcal{J}(t) \), a journey for demand \((R_j, f_j) \in \mathcal{D}(t) \) is a tuple \((A_j(t), \delta_j(t))\), where \( \delta_j \in \mathbb{R} \) is the departure time and is a decision variable of the scheduler, and \( A_j(t) \) is the realized set of arrival times of the UAV at nodes along \( R_j \) that have occurred by time \( t \). Here, \( \delta_j \) is a decision variable of the scheduler and \( A_j \) is a result of realized travel times. In particular, when \( R_j = \{1_1, 2_1, \ldots, k_R\}, A_j(t) = \{A_{j, l}(t)\}_{l=1}^{k_R} \), where \( A_{j,l}(t) \) is the arrival time at node \( l \) if the arrival has occurred by time \( t \), and \( A_{j,l}(t) = \infty \) if demand \( j \) has not arrived at...
node $\ell$ by time $t$. Therefore, the departure time of the UAV from node $\ell$ is $\delta_{j,\ell} = A_{j,\ell} + w$ where we drop the explicit dependence on time since the equation is understood to hold only for times $t$ such that $A_{j,\ell}(t) < \infty$ and we let $\delta_{j,0} = \delta_j$.

For safety reasons, it is assumed that a UAV must be able to land immediately upon arrival at any node along its route. Whenever a UAV arrives at a node along the route of a journey, the journey will be updated accordingly, so that the uncertainty of the rest of the journey decreases. In particular, suppose the UAV departs some node $\ell_1 - 1$ along its route. The latest arrival time at some other node $\ell_2 \geq \ell_1$ along the route is denoted $a_{\ell_1, \ell_2}$ and given by

$$a_{\ell_1, \ell_2} = \delta_{j,\ell_1-1} + \sum_{\ell=\ell_1}^{\ell_2} \tau_{\ell} + (\ell_2 - \ell_1)w,$$

i.e., $a_{\ell_1, \ell_2}$ is the departure time from node $\ell_1 - 1$ plus the upper bound of the time interval it takes to travel through the links $\{\ell\}_{\ell=\ell_1}^{\ell_2}$, with the time spent at each node. Note that it is time-dependent because $\delta_{j,\ell_1-1}$ is updated over time. Further, the time interval that the UAV will potentially block a landing spot at node $\ell_2$ when departing from $\ell_1$ is given by

$$M_{\ell_1, \ell_2} = \left[\delta_{j,\ell_1-1} + \sum_{\ell=\ell_1}^{\ell_2} \tau_{\ell} + (\ell_2 - \ell_1)w, a_{\ell_1, \ell_2} + w\right].$$

We also define $m_{\ell_1, \ell_2}$ as the length of the time interval above, so that

$$m_{\ell_1, \ell_2} = \sum_{\ell=\ell_1}^{\ell_2} \tau_{\ell} - \sum_{\ell=\ell_1}^{\ell_2} \tau_{\ell} + w.$$  

Note that the superscript of $m$ is $R_j$ because it depends only on the route of demand $j$ and not its particular arrival and departure times. We let $M_{v_1, v_2} = M_{\ell_1, \ell_2}$ if $v_1, v_2$ are two nodes along the route $R_j \in R_j$, $v_1 = \ell_{R_j}, v_2 = \ell_{R_j}$ and $\ell_1 \leq \ell_2$. For all $\ell \in \{1, \ldots, k_{R_j}\}$, we let $a_0^{\ell} = a_1^{\ell}, M_0^{\ell} = M_1^{\ell}$ and $m_0^{\ell} = m_1^{\ell}$. In the same manner, we let $a_v^{\ell} = a_v^{\ell}, M_v^{\ell} = M_v^{\ell}$ and $m_v^{\ell} = m_v^{\ell}$ if $v = \ell_{R_j}$.

The task, then, is to assign to each demand a journey such that capacity constraints and arrival times are satisfied.

**Definition 2 (Schedules and Journeys):** Given a set of demands $D(t_0) = \{(R_j, f_j)\}_{j \in J(t_0)}$ at time $t_0$, a corresponding set of departure times $S(t_0) = \{\delta_j\}_{j \in J(t_0)}$ with $J(t_0) \subseteq J(t)$ and each $\delta_j \in R$ is a schedule for $D(t_0)$ if:

1) The latest arrival time is not later than the deadline for all demands $j \in J(t_0)$, that is, $a_{k_{R_j}}^j \leq f_j$, where we recall that $k_{R_j}$ is the final destination node for demand $j$ along route $R_j$ and $a_{k_{R_j}}^j$ is computed as in (1) which depends on any realized arrival/departure times at intermediate nodes;

2) The number of vehicles at a node never exceeds capacity, i.e., for all $v \in V$ and all $t \geq 0$,

$$\sum_{j:v \in V(R_j)} 1(t; M_j^v(t_0)) \leq C_v$$

where the notation $1(\cdot; \cdot)$ is an indicator such that $1(t; [a,b]) = 1$ if $t \in [a,b]$ and $1(t; [a,b]) = 0$ otherwise; and

3) In any finite time interval, only a finite number of UAVs depart.

A schedule is a complete schedule if $\hat{J}(t_0) = J(t_0)$; otherwise it is a partial schedule. For any scheduled demand $j \in \hat{J}(t_0)$, the pair $(A_j(t), \delta_j)$ is the journey of demand $j$ where $A_j(t)$ is the set of realized arrival times as defined above.

In the above definition, items 1 and 2 are the main features of a schedule, while item 3 is a mild technical requirement that is always satisfied in practice. In this paper, we consider the following problem.

**Scheduling Problem.** Given a set of demands $D(t)$ and a sequence of scheduling times $T = \{t_0,t_1,t_2,\ldots\}$ that may be finite or infinite satisfying $t_i < t_{i+1}$ for all $i$, at each scheduling time $t_i$, for the set of available demands $D(t_i)$, determine $\hat{J}(t_i) \subseteq J(t_i)$ the subset of flights to be scheduled and compute a schedule $S(t_i) = \{\delta_j\}_{j \in \hat{J}(t_i)}$ satisfying the properties

1) $\hat{J}(t_i) \supseteq \hat{J}(t_{i-1})$ and $S(t_i) = S(t_{i-1}) \cup \{\delta_j\}_{j \in \hat{J}(t_i) \setminus \hat{J}(t_{i-1})}$, and

2) For all $j \in \hat{J}(t_i) \setminus \hat{J}(t_{i-1})$, $\delta_j \geq t_i$ with $\hat{J}(t_{i-1}) := \emptyset$ and $S(t_{i-1}) := \emptyset$.

Property 1 of the above problem statement implies that once a demand is scheduled for departure, its departure time does not change at future scheduling times. Property 2 captures a causality requirement and implies that newly scheduled demands cannot have departure times in the past.

If $D$ is time-invariant and only a single scheduling time $t_0 = 0$ is considered, the scheduling problem is called static. This was the focus in our preliminary work [12]. Otherwise, it is called a dynamic scheduling problem.

Dynamic scheduling allows the schedule to be updated dynamically at certain scheduling times. Schedule updates may be needed to accommodate new demands or because it may not be possible to schedule all known demands at some time, i.e., only a partial schedule can be obtained. Then, at future scheduling times after some flights arrive at intermediate nodes and reduce uncertainty in the remaining travel time, these previously unscheduled demands can be scheduled. Note that scheduling times need not be fixed in advance and can, for example, be triggered when a flight lands at an intermediate node.

We demonstrate how, even with a fixed and known set of demands, dynamic scheduling may be beneficial.

![Two-link network](image)

**Example 1:** Consider a network with 3 nodes and 2 links as shown in Figure 1. This network has a single route $R = \{e_1,e_2\}$. Suppose the travel time interval of link $e_1$ is $[1,4]$ and of link $e_2$ is $[2,3]$, the wait time at nodes $v_2$ and $v_3$ is $w = 1$, and the capacity of nodes $v_2$ and $v_3$ are $C_2 = C_3 = 1$. Consider the time-invariant set of demands $D =$
A necessary condition for dynamic scheduling of additional demands

Consider the general network setting defined in Section II. Given a set of demands $D(t)$ and a sequence of increasing scheduling times $t_0 < t_1 < t_2 < \ldots$, we consider the set of available demands $D(t_i) = \{(R_j, f_j)\}_{j \in \mathcal{J}(t_i)}$ and a schedule $S(t_i) = \{\delta_j\}_{j \in \mathcal{J}(t_i)}$ for any $i = 0, 1, \ldots$, where $\mathcal{J}(t_i) \subseteq \mathcal{J}(t_{i-1})$ and let $\mathcal{J}(t_{i-1}) = \emptyset$.

We first introduce a cumulative departure function in time interval $[t_1, t_2]$ as

\[ \Delta_v^R(t_1, t_2, \mathcal{J}) : \mathbb{R}^2 \to \mathbb{N} \]

for all $v \in \mathcal{V} \setminus T$, where we recall the set of tail nodes $T$, so that $\Delta_v^R(t_1, t_2, \mathcal{J})$ is the number of UAVs departing from node $v$ in the time interval $[t_1, t_2]$, while traveling through route $R \in \mathcal{R}$, i.e.,

\[ \Delta_v^R(t_1, t_2, \mathcal{J}) = \{ j \in \mathcal{J} | R_j = R \text{ and } \delta_{j,v} \in [t_1, t_2] \}. \]

We then introduce a cumulative arrival function, $\Gamma_v^R(t_1, t_2, \mathcal{J}) : \mathbb{R}^2 \to \mathbb{N}$ as the cumulative number of UAVs that travel through route $R \in \mathcal{R}$ and must arrive at the node $v \in \mathcal{V}(\mathcal{R}) \setminus \{0^R\}$ in the time interval $[t_1, t_2]$, i.e.,

\[ \Gamma_v^R(t_1, t_2, \mathcal{J}) = \{ j \in \mathcal{J} | R_j = R \text{ and } \mathcal{M}_v \subseteq [t_1, t_2] \}. \]

We then define the flow rate at node $v$ through route $R$ in a finite interval $[t_1, t_2]$ as

\[ \tilde{r}_v^R(t_1, t_2, \mathcal{J}) = \frac{\Delta_v^R(t_1, t_2, \mathcal{J})}{t_2 - t_1}, \]

where $v \in \mathcal{V} \setminus T$ and $R \in \mathcal{R}$.

Before exploring the necessary conditions of the schedules and demands, we need to define the bottleneck of a network, which is a basis of the necessary conditions and the algorithms.

**Definition 3 (s-t Cut):** An s-t cut $C = (S', T')$ is a partition of $\mathcal{V}$ such that $s \in S'$ and $t' \in T'$, while $S \subseteq S'$ and $T \subseteq T'$.

For an s-t cut $C = (S', T')$ of the graph $\mathcal{G}$, we further define the set of edges with tails in $S'$ and heads in $T'$ as adjacent edges of the cut $C$, $E_{S' \to T'} = \{ e \in E : \tau(e) \in S' \text{ and } \sigma(e) \in T' \}$. We let $\tau(E_{S' \to T'})$ (resp., $\sigma(E_{S' \to T'})$) be the set of vertices that are tails (resp., heads) of all adjacency edges.

**Definition 4 (Flow and Maximizing Flow):** A set $\{\tilde{r}_v^R\}_{v \in \mathcal{V}, R \in \mathcal{R}}$ with each $\tilde{r}_v^R \in \mathbb{R}_+$ is a flow if the following are satisfied:

\[ \tilde{r}_v^R \geq 0 \quad \forall R \in \mathcal{R}, \quad v \in \mathcal{V} \tag{7} \]
\[ \tilde{r}_v^R = 0 \quad \forall R \in \mathcal{R}, \quad v \in \mathcal{V} \setminus \mathcal{V}(\mathcal{R}) \tag{8} \]
\[ \tilde{r}_v^R \leq \tilde{r}_v^R \quad \forall R \in \mathcal{R}, \quad v \in \mathcal{V} \tag{9} \]

\[ \sum_{R \in \mathcal{R}} \tilde{r}_v^R m_{v-1} \leq C_v \quad \forall v \in \mathcal{V} \setminus S \tag{10} \]

where $r_{i+1}^R = C_{i+1}/m_{i+1}$, with $m_{i+1}$ as defined in (3) and we let $\tilde{r}_v^R = r_{i+1}^R$. Then $\tilde{r}_v^R$ is the maximum flow rate through the node $v$ along the route $R$.

A flow is a maximizing flow if it maximizes $\sum_{R \in \mathcal{R}} \tilde{r}_v^R$ over all flows.

**Definition 5 (Bottleneck):** The bottleneck of a UAM network $\mathcal{N} = (\mathcal{G}, C, \mathcal{R}, \mathcal{F}, \mathcal{M}, w)$ is a set of nodes $\mathcal{V}$ such that for all $R \in \mathcal{R}$, there exists a node $v \in \mathcal{V}$ and a maximizing flow $\{\tilde{r}_v^R\}_{v \in \mathcal{V}, R \in \mathcal{R}}$ such that $\tilde{r}_v^R = r_v^R$ and that in the reduced graph $\mathcal{G}_{\text{reduced}} = (\mathcal{V}, \mathcal{E}_{\text{reduced}}(\mathcal{V}, \mathcal{F}, \mathcal{M}, w))$, for any $v_1 \in S$ and $v_2 \in T$, $v_1$ and $v_2$ are not connected, where $\mathcal{E}_1 = \{ e \in \mathcal{E} | \tau(e) \in \mathcal{V} \}$ and $\mathcal{E}_2 = \{ e \in \mathcal{E} | \sigma(e) \in \mathcal{V} \}$.

The lemma below provides an upper bound on the number of UAVs traveling through a node $v \in \mathcal{V} \setminus S$ that may be newly assigned in a schedule at any scheduling time.

**Lemma 1:** Consider a network $\mathcal{N} = (\mathcal{G}, C, \mathcal{R}, \mathcal{F}, \mathcal{M}, w)$ where $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a sequence of increasing scheduling times $t_0 < t_1 < t_2 < \ldots$. Given the set of demands $D(t)$ and
Consider a network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the set of nodes $\mathcal{V}_s$ consists of a central node $v_0$ and $L$ leaf nodes $v_l$ for $l = 1, 2, \ldots, L$. Then $\mathcal{E}_s = \{(v_i, v_0) \mid 1 \leq l \leq L\}$ is the set of links for the network. We label edge $(v_i, v_0)$ simply as edge $i$. We consider there to be $L$ routes in $\mathcal{R}$ and each route only consists of one edge. We call such a network a local network because it can be interpreted as a local portion of a larger network where we study only incoming flights to a particular node. Consider the static set of demands $\mathcal{D} = (R_j, f_j)_{j \in \mathcal{J}}$. We then define the collection of demands $\mathcal{D}$ as feasible if there exists a schedule for it. If $\mathcal{D}$ is a finite collection of demands, then there always exists a schedule since departure times may be scheduled as early as needed to satisfy capacity constraints and desired arrival times. When $\mathcal{D}$ is an infinite collection of demands, then $\mathcal{D}$ may or may not be feasible. We assume that the number of demands within the time interval $[\tau, \tau + T]$ is finite for all $T < \infty$ and the limiting average number of deadlines in $[\tau, \tau + T]$ as $T \to \infty$ exists and is constant for varying $\tau \in \mathcal{R}$, e.g., $\mathcal{D}$ is a periodic set of demands. We next explore the fundamental limitations on the feasibility of the infinite set of demands $\mathcal{D}$ in the remainder of this section.

We introduce a cumulative departure function in time interval $[t_1, t_2]$ as $\Delta_v(t_1, t_2) : \mathbb{R}^2 \to \mathbb{N}$ for all $v \in \mathcal{V} \setminus \{v_0\}$ so that $\Delta_v(t_1, t_2)$ is the number of UAVs departing from node $v$ in the time interval $[t_1, t_2]$, i.e.,

$$\Delta_v(t_1, t_2) = |\{j \in \mathcal{J} \mid o_j = v \text{ and } \delta_j \in [t_1, t_2]\}|.$$

Given the cumulative departure function, we define the long-term average departure rate $r_v$ at node $v$ in $\mathcal{V} \setminus \{v_0\}$ as

$$r_v := \lim_{T \to +\infty} \frac{1}{T} \Delta_v(\tau, \tau + T), \quad \forall \tau \in \mathbb{R}. \quad (17)$$
In the same manner, we introduce a cumulative arrival function \( \Gamma_v(t_1, t_2) : \mathbb{R}^2 \to \mathbb{N} \) as the cumulative number of UAVs that depart from origin \( v \) and arrive at the destination in the time interval \([t_1, t_2] \), i.e., \( \Gamma_v(t_1, t_2) = | \{ j \in J | o_j = v \text{ and vehicle arrives in } [t_1, t_2] \} | \). In [12, Lemma 1], it was established that for a local network, \( \lim_{T \to +\infty} \frac{1}{T} \Gamma_v(\tau, \tau + T) = r_v \) for all \( \tau \in \mathbb{R} \).

In the following theorem, we obtain a necessary and sufficient condition for the existence of a feasible schedule for a local network when the set of demands \( D \) is infinitely large, so that we can say immediately if there exists any feasible schedule. This will hence provide fundamental limits for how large demands a local network can handle.

**Theorem 2:** Consider a local UAM network \( \mathcal{N} \). An infinite set of demands \( D \) is feasible if and only if
\[
\sum_{1 \leq i \leq L} r_{v_i} \cdot (\tau_i - \tau_0 + w) \leq C_{v_0},
\]
where \( r_{v_i} \) is as given in (17) for all \( v_i \in V \setminus \{v_0\} \) and we recall \( C_{v_0} \) is the capacity of the destination node \( v_0 \).

The proof of Theorem 2 is omitted but follows from the two lemmas below, which consider the special case when \( C_{v_0} = 1 \).

**Lemma 2:** Consider a local UAM network \( \mathcal{N} \) with \( C_{v_0} = 1 \). If an infinite set of demands \( D \) is feasible, then
\[
\sum_{1 \leq i \leq L} r_{v_i} \cdot (\tau_i - \tau_0 + w) \leq 1,
\]
where \( r_{v_i} \) is as given in (17) for all \( v_i \in V \setminus \{v_0\} \).

**Lemma 3:** Consider a local UAM network \( \mathcal{N} \) with \( C_{v_0} = 1 \) and a countably infinite set of demands \( D = \{ (o_j, f_j) \}_{j \in J} \) with \( f_j > t_0 \) for all \( j \in J = \mathbb{N} \). If \( \sum_{1 \leq i \leq L} r_{v_i} \cdot (\tau_i - \tau_0 + w) \leq 1 \), then there always exists \( M \in \mathbb{R} \) such that
\[
\sum_{1 \leq i \leq L} \Gamma_{v_i}(t_0, t_0 + T) \cdot (\tau_i - \tau_0 + w) - f_T \leq M
\]
for any \( T > 0 \), where \( f_T = \max_{j \in J \setminus \{f_j \leq T\}} \{ f_j \} \) is the last deadline that needs to be achieved before \( T \). In particular, this implies that the set of demands \( D \) is feasible.

The result [12, Corollary 1] in our preliminary work is a slight generalization of Theorem 2 to the case with intermediate nodes along each branch of the star graph. Further, by considering a star sub-network consisting of a node and its neighbors, Theorem 2 immediately leads to necessary conditions for feasibility of the infinity set of demands for networks with a general graph topology.

**IV. EFFICIENT ALGORITHMS FOR DYNAMIC SCHEDULING**

In this section, we present an algorithm for dynamic scheduling and, as a special case, static scheduling with finite demands. At its core, the algorithm creates a schedule at each scheduling time using branch-and-bound heuristics to efficiently determine candidate orderings for departure times and then a linear program to determine optimal departure times from the set of fixed orderings. We divide the algorithm into four algorithm blocks. The outermost block, Algorithm \( \square \) creates schedules \( S(t_i) \) at scheduling times \( T = \{ t_0, t_1, t_2, \ldots \} \) given a UAM network \( \mathcal{N} \), time-varying demands \( D(t) \), and UAV arrival times \( A(t) \) that are updated according to the description in Section \( \square \). We introduce \( C_j = \{ c^j_t \}_{t=1}^{K_j} \), as the set of parking spots occupied by the UAV of the \( j \)th journey along its route, where \( c^j_t \in \{ 1, \ldots, C_{p_n} \} \) is the parking spot at node \( \ell \) that the UAV occupies upon its arrival. Therefore \( C(t) \in \{ C_j \}_{j \in J(t)} \) is updated while scheduling. The scheduling times \( T \) correspond to when new demands become available to the scheduler or when a UAV lands at a node along its journey, and these times are generally not known in advance. In this way, Algorithm \( \square \) acts as an event-triggered algorithm that creates a new schedule each time a vehicle lands or a new demand is introduced. For computational considerations, Algorithm 1 also allows for considering only the first \( K_0 \) UAV flights with the earliest deadline, and other available demands are scheduled at later scheduling times. In practice, we found that including such a ceiling significantly increases computational speed with negligible effect on the final schedules for all demands.

It is possible that a complete schedule accommodating worst case travel times may not be possible at each scheduling event. In this case, Algorithm \( \square \) creates a partial schedule. The unassigned demands are then considered at the next scheduling time, and any demand that cannot be fulfilled anymore is ultimately dropped.
Algorithm 2 Schedule-Computing Algorithm

1: function INSERTION(D, Soold, t, A, C, N)
2:  Inputs: demands D = {D_j}j∈J_old, existing schedule \{S_j\}j∈J old := Soold, time t, arrival times A, parking-spot assignments \{(c_{d,v})_{v∈V(R_j)}\}_{j∈J(t)} = C, UAV network ((V,E), C, R, \vec{f}, \vec{w}) := N
3:  \Ell = \arg \min_{t∈V(R_j)} a_{\ell,k,R_j}^j(t) \quad \forall j \in J_old
4:  \Md_{v,c} := \cup_{j∈J_old} a_{\ell,k,R_j}^j(t) \quad \forall v \in V \setminus \delta(D)
5:  \Md := \{\Md_{v,c}\}_{v∈V,c=1,...,C_v}
6:  \M_max := \max_{v∈V,c=1,...,C} (\sup \Md_{v,c})
7:  \mathcal{D}_1 := \{(R_j, f_j) ∈ D \mid f_j − m_{R_j}^{\ell} + w ≤ \M_max\}, sort descending wrt deadline f_j
8:  \mathcal{D}_2 := \mathcal{D} \setminus \mathcal{D}_1, sort ascending wrt f_j
9:  while \mathcal{D}_2 ≠ ∅ do
10:    \mathcal{S} := Soold \cup \\{\delta_j\}; update \Md
11:  for \mathcal{D}_2 \in \mathcal{D}_1 do
12:    go through M and find the latest feasible departure time for D_j which satisfies scheduling constraints at each node along R_j and assign the available spot as c_{d,v} for all v ∈ V(R_j)
13:      if a feasible departure time \delta_j is found then
14:        \mathcal{S} = Soold \cup \\{\delta_j\}; update \Md
15:      else
16:        return Soold \quad \triangleright No feasible schedule found
17:      end if
18:  end for
19:  \mathcal{S}_2 := PREPARE_SCHEDULE(\mathcal{D}_2)
20:  if \mathcal{S}_2 ≠ ∅ then
21:    \mathcal{J}_a := index set of \mathcal{D}_2
22:    \mathcal{J}_f := \{max_{j∈\mathcal{J}_a} f_j + w, \forall v \in V, 1 ≤ c ≤ C_v\}
23:  for \mathcal{J}_a ≠ ∅ do
24:    ﬂ := \arg \max_{j∈\mathcal{J}_a} sup \mathcal{M}_R_j^{f,c}(t)
25:    \c_{d,v} := \max_{c=1,...,C_v} \{fl, c_{d,v}\}, \forall v ∈ V(R_j)
26:    \mathcal{J}_a := \mathcal{J}_a \setminus \{\fl\}
27:  end for
28:  \mathcal{S} = \mathcal{S} ∪ \mathcal{S}_2
29:  return \mathcal{S}
30:  else
31:    remove the first journey D_j ∈ \mathcal{D}_2 and append to the head of \mathcal{D}_1
32:  end if
33:  end while
34:  return Soold \quad \triangleright no feasible schedule found
35: end function

Algorithm 2 seeks a feasible partial or complete schedule minimizing the Sum of Difference (SoD) cost

SoD = \sum_{j∈\hat{J}} (f_j − \delta_j) \tag{21}

where \hat{J} is the index set for the schedule. We refer to this cost as the SoD cost since it is the sum of the difference between arrival time and departure time for the flights. This cost corresponds to the typical preferences of customers to leave no earlier than necessary while still guaranteeing arrival by a desired deadline. An immediate lower bound for the SoD cost is obtained by considering worst case travel times along the routes for all demands, \hat{J}, ignoring capacity constraints at nodes.

With relatively few demands and a simple network topology, e.g., a star network as considered in [12], the exact optimal schedule minimizing \eqref{eq:21} can be obtained from a mixed-integer program. However, for general acyclic directed graphs and/or a large set of demands, obtaining a schedule from a mixed-integer program is not computationally tractable. We next propose an efficient but possibly suboptimal approach for these cases in Algorithm 3.

Algorithm 3 considers a set of demands that need to be scheduled and partitions them into two sets according to their deadlines and existing schedule. To achieve this, the algorithm first computes the time intervals that UAVs will potentially occupy each parking spot of each node according to \eqref{eq:22} and the parking spot assignment C(t). This is time-varying since it depends on time-varying arrivals A_t(t), and we let \Md(t) collect all of the blocked time intervals computed by \eqref{eq:22} for all journeys at all parking spots of all nodes at time t. When a UAV lands at a node along its route, the time interval that the UAV will potentially occupy any node along the rest of its route reduces since the uncertainty of traveling is reduced, so that we can update A(t) and \M(t) accordingly. When journeys are assigned to new demands at time t, \M(t) is used to avoid parking spot conflicts. Using \M(t), Algorithm 2 divides demands into those with earlier deadlines, \mathcal{D}_1, which are inserted into the current schedule if possible while the demands with later deadlines, \mathcal{D}_2, are scheduled using Algorithm 3 beyond the latest time of the existing schedule. If a feasible complete schedule for \mathcal{D}_2 cannot be found, the demand with the earliest deadline is moved from \mathcal{D}_2 to \mathcal{D}_1 and the process repeats.

Algorithms 3 and 4 are presented in the Appendix with MATLAB-like syntax. Algorithm 3 prepares the variables for Algorithm 4, which picks the best solution returned from Algorithm 4 and transforms that into a schedule. Algorithm 4 is a branch-and-bound algorithm that finds a list of possible schedules for the given demands. While the algorithm is inspired by the classical Bratley’s algorithm for task scheduling [22], several adoptions have been made for the problem we are addressing. For instance, Bratley’s algorithm generally seeks a single schedule with the earliest possible departure times, while Algorithm 4 returns a set of possible departure times aiming to minimize the SoD cost \eqref{eq:21}.

Algorithm 4 also employs a pruning technique different than Bratley’s algorithm. Consider the set of candidate schedules as a rooted tree. The algorithm starts from scheduling the last journey for the unassigned demands by visiting the demands in descending order according to their deadlines, picking a demand, recording the latest possible departure time and removing the demand from the unassigned list. The process is repeated until all demands are assigned or the branch is stopped by the pruning mechanism. If all demands are assigned, the current branch will be stored as a possible schedule and the algorithm will continue with another branch.
until all branches are considered (visited or pruned). The algorithm then returns all stored schedules.

We suggest several pruning mechanisms to avoid an exhaustive search, with the first two inspired by Bratley’s Algorithm:

1. If an unassigned demand cannot be scheduled according to the assigned schedules without considering other unassigned demands, the current branch will be discarded.
2. If any node cannot afford the unassigned demands according to Lemma 1 with the assigned schedules, the current branch will be pruned.
3. If the demand with latest deadline can be scheduled without interfering with any other demand, then it is scheduled as the last journey to arrive at the destination.
4. If two demands choose the same route, their order follows the order of their deadline, i.e., if \( D_{j_1}, D_{j_2} \in D_m \) and \( R_{j_1} = R_{j_2} \), while \( f_{j_1} < f_{j_2} \), then only the branches with \( D_{j_1} \) arriving earlier then \( D_{j_2} \) will be preserved.
5. When searching along a branch, the current scheduled demands are compared with stored schedules. If the SoD cost along the current branch must exceed that of any of the stored schedules, the current branch will be abandoned.

Even with the above pruning techniques, Algorithm 4 may still produce a large set of feasible schedules. Although there can be several branches to explore, it is possible to stop the search at any point, given that at least one branch has been found, to get a timely but sub-optimal solution.

### V. Numerical Study

In this section, we demonstrate our algorithm on two case studies with up to 200 UAVs each. We first demonstrate the algorithm on an eight-node network with dynamic scheduling updates. In the second case study, we consider a static scheduling example for a network of the Atlanta region that appeared in our prior work [12]. For this case, we show that the proposed algorithm generates a schedule with a nearly optimal cost in considerably less computational time than the exact integer program proposed in [12]. In both case studies, the proposed algorithm obtained reasonable schedules within 1 second.

#### A. Case Study 1: Dynamic Scheduling

![Fig. 2. A network with 8 nodes and 8 links that is used to illustrate the case study in V-A](image)

The first case study considers the network in Fig. 2. The set of all possible origins (resp., destinations) is \( S = \{v_1, v_3\} \) (resp., \( T = \{v_7, v_8\} \)). We assume the origins \( v_1, v_3 \) do not have capacity constraints, while \( C_{v_2} = 1, C_{v_4} = 2, C_{v_5} = 3, C_{v_6} = 1, C_{v_7} = 3 \) and \( C_{v_8} = 5 \). The links are indicated in the figure and the corresponding travel time intervals are labeled beside the links. We consider four routes \( R = \{R^1, R^2, R^3, R^4\} \) with \( R^1 = \{(v_1, v_2), (v_2, v_4), (v_4, v_7)\}, R^2 = \{(v_1, v_2), (v_2, v_5), (v_5, v_8)\}, R^3 = \{(v_2, v_5), (v_5, v_8)\}\) and \( R^4 = \{(v_3, v_5), (v_5, v_8)\}\). Each UAV remains at the vertistops along its path for \( w = 1 \) minute after landing (all times are given in minutes). Algorithms 1, 2, 3 and 4 are implemented in MATLAB to obtain a schedule \( S = \{\delta_j\}_{j \in J} \).

In simulation, realized travel times are uniformly randomly drawn from the travel time intervals. We first consider 43 randomly generated demands. Subsets of demands become available for scheduling across ten scheduling times \( T = \{0, 15, 30, 45, 50, 60, 80, 85, 100, 120\} \).

Fig. 3 demonstrates the resulting schedule. The figure compares the scheduled and actual arrivals at node 8 in the time interval [100, 160] minutes as observed at time \( t = 100 \) minutes (top) and \( t = 120 \) minutes (bottom). The green, orange and blue bars represent the reserved time slots for UAVs traveling through \( R^2, R^3 \) and \( R^4 \) respectively. UAVs on route \( R^1 \) are not shown because \( R^1 \) does not go through node 8. We label an ID number above each bar to identify the UAVs. The diamond represents the arrival deadline of a demand; the solid-color bar represents the time interval that the UAV could possibly arrive given the schedule and uncertain travel times. The UAV will then stay at the node for time \( w = 1 \) after arrival, which is represented by the lightly shaded bars in the figure. Therefore, the entire bar represents the time interval the UAV may appear at node 8 under best and worst case travel times. Comparing the top and the bottom figures, we observe several representative changes; the journey 30 has not yet arrived at destination by time 100 in the top plot but is completed by time 120 in the bottom plot; the time slot reserved for journey 25 has shrunk because the UAV arrived at an intermediate node along its route in the time interval [100, 120] minute so that the uncertainty of its arrival at node 8 reduces; and journeys 37–43 are newly scheduled in the bottom plot.

A schedule for the 43 UAVs described above is obtained in 0.8 seconds with a SoD cost of 759.3 minutes. By considering worst case travel times for all demands but not capacity constraints, the lower bound for the SoD cost is 746 minutes. Thus, the algorithm produces a schedule that is at least within 2% of optimal, and since an exact optimal schedule is not computationally tractable, it is unknown exactly how far from optimal the obtained scheduled is, or even if it is, in fact, optimal itself.

Next, we consider a larger set of demands with size 200. The routes are chosen randomly among the four routes and the deadlines are picked randomly in the interval [40, 1540] minutes. The algorithm finds its first feasible schedule at 0.2801 seconds with SoD cost equal to 3759.9 minutes, which is again within 2% of the lower bound of 3693 minutes.
B. Case Study 2: Atlanta Network

We next consider the network shown in Fig. 4 that appears also in [12]. This network is inspired by a recent report by INRIX [6] which suggests that a UAM local network traveling between Atlanta node and the three leaf nodes “ALP”, “KEN” and “BUF”, respectively. The time interval need for traveling through each link is labeled beside the corresponding link.

Each UAV stays at the intermediate vertistops along its path for $w_t = 1$ minute and at the Atlanta vertiport for $w = 5$ minutes after landing.

We assume the number of UAV demands across the three origins is $[h_1, h_2, h_3] = [4, 4, 19]$ and that arrival deadlines are set at regular intervals, i.e., if origin $v_i$ is tasked with sending $h_i$ UAVs to Atlanta, the deadlines are $\frac{360}{h_i} \cdot k + 0.5$, $k = 1, 2, \ldots, h_i$.

For comparison, the mixed-integer optimization problem in [12] is solved in MATLAB using Gurobi with YALMIP toolbox to obtain a schedule that exactly minimizes the SoD cost (21). Using the exact mixed-integer program from [12] takes around 150 minutes of computation time on a standard laptop, and the optimal SoD cost as in (21) is 1587 minutes. Note that the lower bound computed using worst cast travel time but ignoring capacity constraints is 1173 minutes. Thus, the capacity constraints at the nodes play a significant role in overall network SoD cost.

In contrast, the algorithm proposed in Section IV obtains a feasible schedule in 0.05 seconds. The SoD cost of this sub-optimal schedule is 1669 minutes. The SoD cost reduces to 1605 minutes when the algorithm finds an alternative schedule after 5.8 seconds of computation time. Fig. 5 demonstrates how the minimal SoD cost of all stored schedules decreases with computation time.

The exact mixed-integer program cannot practically compute a solution if the number of demands exceeds about 50. On the other hand, the scheduling algorithm in this paper provides a sub-optimal schedule quickly for even a large demand set. To demonstrate this, we now consider 200 demands assigned randomly to the three routes with random deadlines in the interval [40, 1540] minutes. The algorithm finds the first feasible schedule at 0.2 seconds with Sum of Difference equal to 12662 minutes and, after 10.1 seconds, obtains a schedule with Sum of Difference equal to 12637 minutes. The lower bound for this set of demands is 7786 minutes. We note, however, that this is only a lower bound and it is likely that the optimal schedule has SoD cost significantly greater than this bound as was the case in the prior example.
with 27 UAVs.

VI. CONCLUSION

In this paper, we studied the problem of dynamic scheduling in UAM networks with uncertain travel time. One main challenge is that nodes in a UAM network have limited parking spaces. As a result, a schedule for each UAV in the network has to be made before it takes off to ensure that a parking space is available upon arrival. Additionally, a mixed integer program is too time-expensive when the complexity of network and the size of demands increase.

An exact schedule can sometimes be obtained from a mixed integer program, but this is not computationally tractable for larger networks and/or large sets of demands. Instead, we present a dynamic scheduling algorithm that is able to consider new demands over time and uses branch-and-bound heuristics to identify feasible but possibly sub-optimal schedules. In addition, we presented theoretical results establishing necessary conditions for a schedule to be feasible, and we further showed that these conditions become also sufficient conditions in certain cases. Future work could consider scheduling for unforeseen disruptions such as one or more UAVs needing to reroute or land due to, e.g., adverse weather conditions.

REFERENCES

[1] J. Holden and N. Guel, “Fast-forwarding to a future of on-demand urban air transportation,” 2016. [Online]. Available: https://www.uber.com/elevate.pdf

[2] D. P. Thipphavong, R. Apaza, B. Barmore, V. Battiste, B. Burian, Q. Dao, M. Feary, S. Go, K. H. Goodrich, J. Homola et al., “Urban air mobility airspace integration concepts and considerations,” in 2018 Aviation Technology, Integration, and Operations Conference, 2018, p. 3676.

[3] K. Balakrishnan, J. Polastre, J. Moolberry, R. Golding, and P. Sachs, “Blueprint for the sky: The roadmap for the safe integration of autonomous aircraft,” Sky, vol. 3, 2018.

[4] “Nextgen independent assessment recommendations,” McLean, VA: The MITRE Corporation, 2014.

[5] B. Cascara, T. Spencer, M. DeGarmo, A. Lacher, D. Maroney, and M. Gutierrez, “Urban air mobility landscape report: Initial examination of a new air transportation system,” McLean, VA: The MITRE Corporation, 2018.

[6] INRIX, “Electric passenger drones could relieve housing costs and spread growth in nation’s booming cities,” 2019. [Online]. Available: https://inrix.com/campaigns/vtol-study/

[7] C. Al Haddad, “Identifying the factors affecting the use and adoption of urban air mobility,” 2018. [Online]. Available: https://mediatum.ub.tum.de/1482026

[8] E. Ancel, F. M. Capristan, J. V. Foster, and R. C. Condotta, “Real-time risk assessment framework for unmanned aircraft system (UAS) traffic management (UTM),” in 17th AIAA Aviation Technology, Integration, and Operations Conference, 2017, p. 3273.

[9] C. Bosson and T. A. Lauderdale, “Simulation evaluations of an autonomous urban air mobility network management and separation service,” in 2018 Aviation Technology, Integration, and Operations Conference, 2018, p. 3365.

[10] M. Xue, J. Rios, J. Silva, Z. Zhu, and A. K. Ishihara, “Fc3: An evaluation tool for low-altitude air traffic operations,” in 2018 Aviation Technology, Integration, and Operations Conference, 2018, p. 3848.

[11] M. A. Aiello, C. Dross, P. Rogers, L. Humphrey, and J. Hamil, “Practical application of SPARK to OpenUxAS,” in Formal Methods – The Next 30 Years. Springer International Publishing, 2019, pp. 751–761.

[12] Q. Wei, G. Nilsson, and S. Coogan, “Scheduling of urban air mobility services with limited landing capacity and uncertain travel times,” in 2021 American Control Conference, 2021.

[13] V. Bulusu, R. Sengupta, E. R. Mueller, and M. Xue, “A throughput based capacity metric for low-altitude airspace,” in 2018 Aviation Technology, Integration, and Operations Conference, 2018, p. 3032.

[14] D. Sun, K. Zhao, Y. Fang, and J. Cui, “Dynamic traffic scheduling and congestion control across data centers based on SDN,” Future Internet, vol. 10, no. 7, 2018.

[15] M. Gendreau, G. Laporte, and R. Séguin, “Stochastic vehicle routing,” European Journal of Operational Research, vol. 88, no. 1, pp. 3–12, 1996.

[16] B. C. Dean, “Algorithms for minimum-cost paths in time-dependent networks with waiting policies,” Networks: An International Journal, vol. 44, no. 1, pp. 41–46, 2004.

[17] S. Samaranyakane, S. Blandin, and A. Bayen, “A tractable class of algorithms for reliable routing in stochastic networks,” Procedia-Social and Behavioral Sciences, vol. 17, pp. 341–363, 2011.

[18] P. Ioannou and F. d. A. A. Vital, “Optimizing combined truck routing and parking based on parking availability prediction,” No. METRANS Project 17-01, Tech. Rep., 2018.

[19] A. Kok, E. W. Hans, J. M. Schuten, and W. H. Zijm, “A dynamic programming heuristic for vehicle routing with time-dependent travel times and required breaks,” Flexible services and manufacturing journal, vol. 22, no. 1-2, pp. 83-108, 2010.

[20] K. Albers and F. Slomka, “Efficient feasibility analysis for real-time systems with edf scheduling,” in Design, Automation and Test in Europe. IEEE, 2005, pp. 492–497.

[21] N. Fisher and S. Barah, “The global feasibility and schedulability of general task models on multiprocessor platforms,” in 19th Euromicro Conference on Real-Time Systems (ECRTS’07). IEEE, 2007, pp. 51–60.

[22] G. C. Buttazzo, Hard real-time computing systems: predictable scheduling algorithms and applications. Springer Science & Business Media, 2011, vol. 24.

APPENDIX

We explain Algorithm 3 and 4 in this appendix. While Algorithm 3 is the main schedule-finding algorithm that provides a set of possible schedules, Algorithm 4 prepares the variables for Algorithm 3 to pick the best solution returned from Algorithm 3 and transforms that into a schedule.

Algorithm 3 first computes \( f_{j,v} \) by (13) for all \( j \in J \) and \( v \in V(R_j) \), based on the set of unassigned demands \( D_0 = \{D_j\}_{j \in J} \). Recall that \( f_{j,v} \) is the latest time for the UAV to arrive at node \( v \) if \( v \neq 0^{R_j} \) and is the latest departure time if \( v = 0^{R_j} \) without passing its deadline \( f_j \). We initialize \( DLL, PT \) and \( RT \) as zero matrices of dimension \( |J| \times |V| \). We let \( DLL(j,v) = f_{j,v}, PT(j,v) = m_{j,v}^{R_j} \) and \( RT(j,v) = \lambda_{j,v}^{R_j} \) for all \( j \) and \( v \in V(R_j) \setminus \{0^{R_j}\} \), and let \( DLL(0^{R_j}) = f_{j,0^{R_j}}, \) so that \( DLL, PT \) and \( RT \) represent
Algorithm 3 Prepare Schedule

1: function PREPARE_SCHEDULE(D₀ = {D_j}j∈J₁)
2: Let DDL, PT, RT be three N₀ by |V| zero matrices.
3: for j ∈ J₁ do
4: for v ∈ R₁ do
5: \( DDL(j, v) := f_{j,v} \) \( \triangleright f_{j,v} \) from (13)
6: \( PT(j, v) := m_{v_j} \) \( \triangleright m_{v_j} \) from (3)
7: \( RT(j, v) := M_j \) \( \triangleright M_j \) from (3)
8: end for
9: end for
10: \( RID := \text{zeros}(N₀, 1) \), \( RdpT := \text{zeros}(N₀, 1) \), \( SID := \emptyset \), \( SdpT := \emptyset \) \( \triangleright \) Initialize
11: \((SID, SdpT) := \text{SCHEDULE}(D₀, SID, SdpT, RID, RdpT, |D₀|, DDL, PT, RT) \)
12: \( S_{\text{new}} := \emptyset \)
13: if \( SdpT \neq \emptyset \) then
14: \( i^* := \arg \max_i \sum_j SdpT(j, i) \) \( \triangleright i^* \) index of the optimal schedule
15: Let \( S_{\text{new}} \) be the new schedule, with departure times \( SdpT(:, i^*) \) ordered as \( SID(:, i^*) \)
16: end if
17: return \( S_{\text{new}} \)
18: end function

the deadline for arrival/departure, possible time for occupying the nodes, and the shortest time for the UAV to travel from origin to node v along its route, respectively. It then calls the function \textit{scedule} in Algorithm 4 and picks the schedule that achieves the smallest SoD cost.

Algorithm 4 is a branch-and-bound algorithm that considers the set of candidate schedules as a rooted tree. Given the unassigned demands with ascending deadlines \( D₀ = \{D_j\}j∈J₁ \), the algorithm visits the unassigned demands in descending order according to \( f_j \) to pick the last journey in the schedule. \( RdpT \) records the departure time while \( RID \) records the index of the picked demand as in line 19.

We then remove the demand from the unassigned list and delete the row of the corresponding demand from \( DDL, PT \) and \( RT \) before continuing with the current branch. If the set of demands \( D₁ = \{D_j\}j∈J₁ \), where \( J₁ ⊆ J₀ \), is already scheduled along the current branch, then for each node v, we can then compute \( f_{v,c} \) as in Algorithm 2 line 21-23 for all \( v ∈ V \) and \( c = 1, \ldots, C_v \), by substituting \( J₀ = J₁ \) with line 21. We implicitly assume that we will assign the journey to a parking spot where the earliest arrival time of the next journey at the same spot will be the latest among the \( C_v \) parking spots for all assigned demands. We let \( f_{v,c} = \max_{1 ≤ c ≤ C_v} f_{v,c} \) for all \( v ∈ V \) and let \( f_{j,v} = \min(DDL(j,v), f_{v,c}) \) for all \( j ∈ J₀ \setminus J₁ \) and all \( v ∈ V(R_j) \). We then update \( DDL \) so that

\[
DDL(j, 0^{R_j}) = \min_{v ∈ V(R_j) \setminus \{0\}} \{f_{j,v} - M_j\} 
\]  
and

\[
DDL(j, v) = DDL(j, 0^{R_j}) + M_j. 
\]  

The search-and-assign process is repeated until all demands are assigned or the branch is stopped by the pruning mechanism, which will be described later. If all demands are assigned, then the current branch, \( RID \) and \( RdpT \), will be stored into \( SID \) and \( SdpT \) as in line 14. The algorithm will then continue with another branch until all branches are considered (visited or pruned). The algorithm then returns \( SID \) and \( SdpT \).

As mentioned in Section IV, we have five main pruning conditions, while the first, second and the fourth conditions are easy to be realized by line 7, 9 and 36 respectively. We next explain the third and the fifth conditions in detail.

The third condition indicates that if the demand with latest deadline can be scheduled without interfering with any other demand, then we schedule it as the last journey to arrive at the destination. The following explains line 17 in Algorithm 4. Let \( j₀ = \arg \max_j [J₀ \setminus J₁] f_j \). For each \( v ∈ V(R_{j₀}) \), the number of unassigned demands that need to be considered is

\[
num = \min \left( C_v, \left| \{D_j \mid j ∈ J₀ \setminus J₁, v ∈ V(R_{j₀})\} \right| - 1 \right). 
\]  

By our parking-spot assigning assumption, if assigning \( D_{j₀} \) as the last journey among all the unassigned demands will not interfere with any other demand, then

\[
\max^k([f_{v,1}^{node}, \ldots, f_{v,C_v}^{node}]) ≥ \max^{k+1}(DDL(:, v))
\]  
for all \( v ∈ V(R_{j₀}) \) and \( k = 1, \ldots, num \), where we let \( \max^k(\cdot) \) represents the \( k \)th largest number in the vector \( \cdot \).

We now demonstrate how we can realize the last pruning condition with lines 25 and 26. When searching along a branch, we will compare the current scheduled demands with stored assignments in \( SID \) and \( SdpT \). If the expected SoD cost along the current branch already exceeds any of the stored branches, the current branch will be abandoned. Consider the current unassigned demands \( \{D_j\}j∈J₀\setminusJ₁ \). If \( SID(1) = |J₀ \setminus J₁|, col = J₀ \setminus J₁ \) for some \( col ∈ N \), we then compute \( f_{v,c} \) for all \( v ∈ V(R_j) \) and \( c = 1, \ldots, C_v \). Similarly, we can use \( SID(:, col) \) and \( SdpT(:, col) \) to obtain \( \{d_j\}j∈J₁ \) and compute \( f_{v,c}^{node} \) for all \( v ∈ V(R_j) \) and \( c = 1, \ldots, C_v \). If

\[
\max^k([f_{v,1}^{node}, \ldots, f_{v,C_v}^{node}]) = \max^k([f_{v,1}^{node,S}, \ldots, f_{v,C_v}^{node,S}])
\]  
for all \( v ∈ V(R_j) \) and \( k = 1, \ldots, num \), where \( num \) can be computed as in (24), is satisfied, then we can stop searching the current branch and instead use the stored schedule, as initiated on line 26.

If more than one \( col \) schedule satisfies (26), we pick the one with the least SoD cost, \( \text{col}_{\text{min}} \). We let \( SID_2 = SID(1) = |J₀ \setminus J₁|, \text{col}_{\text{min}} \) and \( SdpT_2 = SdpT(1) = |J₀ \setminus J₁|, \text{col}_{\text{min}} \). We then merge \( SID_2, SdpT_2 \) with \( RID, RdpT \) as in line 27-30 and stop searching this branch.
Algorithm 4 Schedule

1: function SCHEDULE(\(D_{in,j}\) \(j \in J\), SID, SdpT, RID, RdpT, N_0, DDL, PT, RT)

2: \(D_{in} = \{D_{in,j} = (R_{in,j}, f_{in,j})\}_{j \in J}\)

3: \(N := |D_{in}|\)

4: \(D = \{D_j = (R_j, f_j)\}_{j=1}^N := \) the sorted sequence of \(D_{in}\), so that \(D = D_{in}\) and \(f_{j_1} \leq f_{j_2}\) if \(j_1 \leq j_2\)

5: \(K := \text{zeros}(N, 1)\)

6: \(K(j) := j'\) if and only if \(D_j = D_{in,j'}\)

7: if \(\exists j, v\) that \(DDL(j, v) < 0\) then

8: return SID, SdpT

9: else if \(\exists v\) that \(\sum_j PT(j, v) < (\max_j DDL(j, v) - \min_j RT(j, v))\) then

10: return SID, SdpT

11: else if \(N = 1\) then

12: \(dpT := DDL(N, 0^{R_i})\)

13: \(RID(1) := K(1), RdpT(1) := dpT\)

14: \(SID := [SID, RID], SdpT := [SdpT, RdpT]\)

15: return SID, SdpT

16: else

17: if \(D_N\) can arrive at the each node at last without requiring any other UAV to depart earlier then \(\triangleright (25)\)

18: \(i := N\)

19: \(RID(N) := K(i), RdpT(N) := DDL(i, 0^{R_i})\)

20: \(D_2 := \{D_j\}_{j \neq i}\)

21: \(DDL_2 := DDL, DDL_2(i, :) := []\)

22: \(PT_2 := PT, PT_2(i, :) := []\)

23: \(RT_2 := RT, RT_2(i, :) := []\)

24: update \(DDL_2\) according to (22) and (23)

25: compare SID, SdpT with \(D_2\)

26: if condition in (26) is satisfied then

27: \(RID(1 : N - 1) := SID_2\)

28: \(RdpT(1 : N - 1) := SdpT_2\)

29: \(SID := [SID, RID]\)

30: \(SdpT := [SdpT, RdpT]\)

31: else

32: \([SID, SdpT] := \text{SCHEDULE}(D_2, SID, SdpT, RID, RdpT, N_0, DDL_2, PT_2, RT_2)\)

33: end if

34: else

35: for \(i \in \{N, \ldots , 1\}\) do

36: if \(\forall j\) such that \(R_j = R_i, f_j \leq f_i\) then

37: same as line [19–33]

38: end if

39: end for

40: end if

41: return SID, SdpT

42: end function