IEEE Open Journal of
Intelligent Transportation
Systems

Received 24 November 2021; revised 15 February 2022 and 21 March 2022; accepted 27 April 2022.
Date of publication 29 April 2022; date of current version 6 May 2022.

Digital Object Identifier 10.1109/OJITS.2022.3171502

Generalized Path Planning for UTM Systems With a Space-Time Graph

RAFAEL PAPA (Graduate Student Member, IEEE), IONUT CARDEI (Senior Member, IEEE), AND MIHAELA CARDEI (Senior Member, IEEE)

Department of Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL 33431, USA

CORRESPONDING AUTHOR: M. CARDEI (e-mail: mcardei@fau.edu)

ABSTRACT Motivated by the increased use of UAS in commercial applications, in this paper we tackle the problem of path planning when requests are submitted by UAS managed by different operators. We propose the new problem of *generalized path planning* for UAS Traffic Management, where the UAS path is described by operators with a sequence of waypoint groups and a solution trajectory must pass through a waypoint in each group. This problem is typical for applications where multiple charging stations and pickup/drop-off locations are distributed in a flight area. Our solution builds upon prior work on discretized space-time graph path planning and proposes a novel multi-source/multi-destination graph search algorithm that generates collision-free trajectories for pre-flight CDR. Our efficient algorithm has runtime proportional to the number of groups and avoids combinatorial explosion. We apply our mechanism to the energy-constrained UAS package delivery problem with multiple warehouses and battery charging stations. Simulation results show that our algorithm is efficient and scalable with the number of requests and graph size. The addition of charging stations and the option for multiple warehouses increases the request admission ratio and reduces the overall trajectory duration, effectively improving both the planner's quality of service and the efficiency of airspace usage.

INDEX TERMS UAS path planning, UTM system, space-time graph, pre-flight CDR.

I. INTRODUCTION

THE UNMANNED Aircraft Systems (UAS) service market is expected to grow to $63.6 billion by 2025 [1]. According to Statista [2], drone sales have surpassed $1.25 billion in 2020. The low cost of acquisition, payload capacity, maneuverability, and the ability to fly at low-altitudes with a very low cost of operation, make unmanned aerial vehicles (UAVs) a perfect fit to revolutionize the payload transportation of small items. There are many commercial applications for UAS at low altitude including package delivery, inspection, surveillance, monitoring and aerial photography.

Unmanned Aerial Vehicle (UAV) technology has evolved considerably in recent years. Many research challenges are still related to the device limitations. Low operating speed, battery lifetime, vulnerability to hackers and little resistance against weather change still affect some real life simulation/results. Flying under rain, snow and/or lighting environments may damage the circuit components and also interfere with the communication between the UAV and the controller.

UAS operations must ensure safety, security, efficiency and equity of the airspace system. The airspace could become congested in the next few years, exceeding the Air Traffic Control (ATC) capacity [3], therefore the need to regulate drone flights and provide a system that ensures a fair distribution of the available airspace to commercial applications has become increasingly important.

To address this challenge, NASA and the U.S. Federal Aviation Administration (FAA) have developed an UAS Traffic Management (UTM) architecture [4] that supports a set of services to enable cooperative management of low-altitude UTM operations between UAS Operators. This architecture supports common situational awareness among

The review of this article was arranged by Associate Editor Fernando Auat Cheein.

TABLE 1. Acronym list.

ATC	Air Traffic Control
BFS	Breadth-First-Search
CDR	Conflict Detection and Resolution
FAA	Federal Aviation Administration
UAS	Unmanned Aircraft System
UAS-ECDS	UAS Energy-constrained Delivery Scheduling
UAV	Unmanned Aerial Vehicle
USS	UAS Service Supplier
UTM	UAS Traffic Management
VTOL	Vertical Takeoff and Landing

all UTM stakeholders (e.g., Operators, FAA, and other government agencies). UTM [4] is a community-based, cooperative traffic management system, where the Operators and entities providing operation support services are responsible for the coordination, execution, and management of operations, with rules of the road established by FAA.

One of the most important components of the UTM architecture is the UAS Service Supplier (or USS). The USS helps UAS operators to meet the UTM operational requirements to provide safety and an efficiency use of the airspace. The USS acts as a communication bridge between the UTM controller and the UAS operators. It provides information about the planned operations within the UTM and it stores operation data in its historical databases for future analysis. Those functions allow the architecture to provide an independent cooperative management of the UAVs operations without the necessity of direct FAA involvement in the process.

UAV path planning is an important operation of the UTM systems. USS must process requests from different UAS operators. Some of the challenges of designing path planning mechanisms are obstacle avoidance (both static and dynamic), dealing with Conflict Detection and Resolution (CDR), scalability with number of requests and graph size, and efficiency. In addition, there are limitations due to UAV technology. For example a commercial drone can only fly for about half an hour in good conditions (i.e., no wind or rain).

Different from a standard aircraft flight plan where its route is propagated through ATC automation systems for aircraft operations, the flight intent (or path planning) in UTM have to be submitted and shared among its Operators in the form of an Operation Plan. The Operation Plan has to be computed in advance and it has to provide the information required by UTM to be able to be approved. For instance, such information could include UAV launch, recovery, planned times, and locations (waypoints) associated with the path. The Operation Plan created by the Operator may be impacted by other planned operations, airspace constraints, or even ground constraints [4]. In such a case, the Operator should be able to adjust the submitted plan accordingly and resubmit its plan for a new evaluation. The UTM system will be responsible to provide all appropriate information affecting the planned operation of an Operator. On the other hand, the operator will need to identify operational conflicts and be able to strategically de-conflict them and the USS will be responsible to provide a fair distribution of the airspace [4].

UAV must avoid static and dynamic obstacles [5]. Static obstacles include airports, stadiums, and other no-fly areas. Dynamic obstacles include UAVs controlled by other service providers. There are two types of CDR: (i) pre-flight CDR performed before the actual flight and it accounts for flights already scheduled, and (ii) in-flight CDR that adapt UAS flight in real-time due to weather or other emergency situations. Research work [5] considers an implementation of the UTM system where Operators use USS for path planning and other tasks, and a central Core UTM to deal with in-flight CDR.

Our main contribution in this article is the formulation of the *generalized path planning* problem for UTM and an algorithm that finds such *collision-free* trajectories with low computational complexity. Our novel algorithm is efficient and scalable in simulations with thousands of vehicles. This algorithm can be used by UTM for pre-flight CDR. It is a generalized solution of the UAS Energy-constrained Delivery Scheduling (UAS-ECDS) problem we introduced in [6]. With *generalized path planning* a UAS operator specifies a path planning request with a sequence of waypoint groups, including a source group and a destination group. A solution trajectory is a *collision-free* sequence of locations that must include one waypoint from each waypoint group in the required order, and that keeps UAS at a safe distance from each other at all times. Generalized path planning is a new problem from all our own prior work and any other prior work in UAS path planning/routing. Traditional path planning is just a special case: when all waypoint groups have size 1, generalized path planning reverts to the traditional waypoint planning problem. Groups with more than one waypoint are practical in scenarios when an operator has access to multiple battery charging stations (or battery resupply/changing stations) and when a UAS can pick up/deliver from/to a group of warehouses or clients. We build upon our prior work on UAS collision-free path planning using a discretized space-time graph ([6], [18], [19]). Our solution is a novel greedy multi-source/multi-destination graph search algorithm that runs on the discretized space-time graph overlayed on the flight volume of operation under UTM control. The algorithm runtime is directly proportional to the product of the number of groups and the maximum waypoint group size and it avoids combinatorial explosion.

The second contribution is a path validation algorithm that checks whether a trajectory supplied by an operator is in conflict with other current or pending trajectories in the same area. The algorithm operates on the discretized space-time graph. Its runtime complexity is directly proportional with the path size or the number of graph vertices. The runtime does not depend in any way with the total number of UAS or trajectories and that makes it very scalable in large scenarios.

For our third contribution we formulate a solution to the UAS Energy-constrained Delivery Scheduling (UAS-ECDS)

problem [6] using the novel generalized path planning algorithm. In the UAS-ECDS problem a UAS picks up a package from any of set of start warehouses and delivers it to a client location. After that, the UAS must fly to any of a set of destination warehouses. The UAS may use optional charging stations as intermediary waypoints to extend their flight range between the source/finish warehouses and the client vertex. A performance evaluation analysis with simulations reveals that our solutions have low runtime, scale well with large networks and deployments, and use graph resources efficiently, with high admission ratio to find tours with low delay.

The rest of the paper is organized as follows. Related works in UAS path planning and our prior work are presented in Section II. Section III discusses the motivation and introduces the problem definition. In Section IV we propose our generalized path planning mechanism, the path validation algorithm, and we apply the former to the UAS-ECDS problem. The performance of our algorithm is discussed in Section V. The conclusions are stated in Section VI.

II. RELATED WORK

UAS path planning is an important research problem that has attracted recently increased attention. In this section we present relevant research in conflict detection and resolution, the NASA UTM architecture concept, and our prior work that serves as foundation for the contributions in this article.

Paper [7] proposes a path planning algorithm that can be used when conventional UAVs and aircrafts need to fly in close proximity. The objective is to generate path trajectories for multiple UAVs and aircrafts flying to multiple destinations while maintaining an adequate separation distance between them. The authors formulate a mixed-integer programming (MIP) problem that can be solved using the Quadratic Unconstrained Binary Optimization (QUBO). Due to their complexity, MIP formulations have high complexity and they are not scalable. To overcome these drawbacks, authors use quantum annealing to solve some of the cases.

The current airspace management system requires a solution that integrates the high-density UAV traffic into the regular airspace traffic. Article [8] extended the aforementioned work by taking into consideration large groups of autonomous UAVs flying close together, i.e., swarms of UAVs. Each swarm is represented by a unique member called "locomotive". The same QUBO optimization problem is used to compute the locomotive path planning and to solve the conflicts between clusters of UAVs. The proposed approach uses instantaneous optimization focusing on local effects, and it does not provide a global optimal path for the entire system.

Paper [9] proposes off-line and on-line approaches for path planning. The off-line approach implements a genetic algorithm (GA) that provides multiples trajectories using the Pareto Front (PF), while the on-line mechanism uses the A* algorithm as a decision factor to prune branches of its existing exploring tree. The PF is used to reduce the number of possible solutions and provides a suitable set to the external operator. The authors have demonstrated that the GA algorithm outperforms the A* when possible solutions are taken into consideration, however the A* mechanism runs faster.

Article [10] computes the path planning using UTM Volumes. Each vehicle has to operate inside one of these approved volumes during the whole operation. The proposed approach uses a tree based local path planner to ensure collision free trajectories of the UAVs inside the UTM approved volumes in high-density urban areas (UTM TCL4). The authors use a recursive tree to generate the paths and then select the one that satisfies all constraints within the approved UTM volume. The local planner is responsible to compute possible trajectories based on a list of waypoints that vehicles have to follow. The internal controller ensures that the UAV will follow a feasible trajectory. The UAV current position is part of a list of local waypoints computed in the previous time step and the current position is used to generate the next one. To avoid collisions, the UAVs are exchanging relevant information through Dedicated Short-Range Communications (DSRC). Simulation results show that the paths computed by the local planer satisfy all the UTM constrains and avoid the static obstacles.

Article [11] designs a path-planning algorithm that combines a flight scheduling heuristic with 4D trajectory optimization. A set of encounter-based metrics are used to estimate the complexity and the capacity of the unmanned airspace. By reducing the fraction of vehicle encounters in the airspace, they can achieve better safety performance and outperform the traditional path planning approaches. This work focuses on gradient-based motion planning algorithms know as trajectory optimization [12] to compute the flight trajectories. Integer linear programming is used to compute the total number of encounters of hypothetical flights in a specific time range. The algorithm mainly works to perform the de-conflicting against other existing trajectories and obstacles in the airspace without considering the capacity constraints and the impact on the current airspace.

Conflict detection and resolution (CDR) is an important aspect to be considered for path planning. Work [5] proposes an adaptation of the optimal reciprocal collision avoidance, a well-known mechanism used in robotics for providing collision-free motion between multiple independent mobile robots. The objective is to minimize the total sum of distances traveled by all UAVs of the fleet using tabu search. Collisions are avoided by selecting the optimal action of each agent based on a low-dimensional linear program resolution. Each agent takes half of the responsibility of avoiding pairwise collisions during a flight. Authors considered scenarios with 3 UAS operators and a fleet varying from 15 to 35 UAVs. However, more scalable approaches are needed for practical applicability.

A different approach is presented by Sacharny *et al.* in [13]. The authors proposed a lane-based method where lanes are created and reserved on demand by USSs, after being approved by UTM authorities. The objective of the

strategic deconfliction algorithm presented in this work is to determine conflicts between UAS flight paths before UAS take off. Authors use a space-time lane diagram where each lane has a single entry point and a single exit point that allows a direct UAS flight between the two ending points. Crossing conflicts are eliminated by an airspace structure inspired by roadway roundabouts, which is a concept from the ground traffic engineering [14]. The innovation of the approach relies on the dynamic nature of lane creation or deletion, as well as on the introduction of virtual roundabouts in airspace to perform strategic deconfliction.

Instead of virtual lanes, article [15] uses geofences to organize the airspace into spatial and temporal segments following the UTM definitions. The main goal is to deconflict new requested geofences from the already approved ones in the UTM system. Several algorithms are introduced for various operations such as adding/removing geofences to the UTM database, managing the geofence temporal bounds, and geofence boundary deconfliction. The evaluation is done in order in which new geofences requests are received and static geofences have higher priority over the dynamic ones.

Performing strategic deconfliction reduces the occurrence of real-time (or tactical) deconfiction during the flight. Several works have introduced communication protocols to overcome the issues created by the tactical deconfliction. For instance, Sacharny *et al.* [16] have used a lane-based model, presented earlier in this section, and have proposed a protocol to coordinate the airspace in the proposed UTM architecture. The protocol is responsible to manage deconfliction using the Closest Point of Approach algorithm and the lane-based infrastructure is responsible to inform the ideal operating density to the UTM environment. The objective is to reduce the speed of one or more UAVs that are heading to the same destination and flying too close from each other.

The UTM architecture presented by NASA is still in the development stage. Paper [17] presents a UTM Core platform used to minimize the impact of multiple USSs connected to the UTM services directly. The objectives are to facilitate data exchange, enable the community-based regulations to be implemented, and use of the airspace more efficiently. The proposed platform centralizes all safety-critical airspace management functions of USSs into a single UTM Core. It also separates the low-altitude functions from the commercial services of UAS operators. The new UTM Core allows the USSs to focus on the main core business and clients while delegating the safety-critical functions to a central operator.

In our prior work from [18] we propose a scalable path planning algorithm that uses an extension of the Breadth-First-Search (BFS) algorithm over a discretized space-time unit graph. The graph representing the aerial highway with airspace available for UAS transit is discretized in segment units equal to the minimum safety distance (or longer). With vehicle speed constrained to 0 m/s (hover) or a uniform cruise speed for all vehicles, the path search algorithm generates conflict-free trajectories that maintain the safety distance between all UAS in operation. An edge pruning mechanism is used to make nearby edges unavailable when computing paths for other vehicles. Some details on the discretized space-time graph and edge pruning are relevant for this article and are presented in Section IV-A. Simulations have been conducted for up to 9000 UAS requests arriving at a rate of 4.5/s with an average run time of 0.152 seconds per request for a map of downtown Boca Raton, Florida, and performance results proved that our space-time graph approach is very scalable for scenarios with thousands of vehicles in the same area. Other UAS and vehicle path planning algorithms only scale with scenarios with much fewer vehicles. We extended this work and article [19] proposes a mechanism that processes the requests in batches, yields better admission ratio and delay overhead in low and medium density scenarios. The running time is 4-7 times shorter compared to the algorithm in Section IV-A, showing even higher scalability in practical situations.

In paper [6] we turned to the problem of UAS path planning for package delivery with multiple warehouses as starting and ending points and with multiple charging stations that can be used to extend UAS operational range. We formulated the UAS Energy-constrained Delivery Scheduling problem and we proposed a path planning mechanism that uses a multi-source A* algorithm variant. The solution assumes a centralized traffic management system that processes all client requests, regardless of their operators, and has global information on the airspace availability. The algorithm proved to be scalable in scenarios with thousands of vehicles operating in the same metropolitan-size airspace.

Our main contribution in this article uses the space-time graph framework from [18] to address a new and different problem that was not considered elsewhere in the UAS/UAV path planning literature, to the best of our knowledge. We propose a novel generalized path planning algorithm which is later applied to solve the UAS-ECDS problem from [6] (see Section IV-D). The proposed generalized path problem takes as input the waypoint groups that must be traversed in sequence and the vertex graph and seeks to compute a path that consists of graph edges that includes exactly one vertex from each waypoint group, in the given order. The algorithm takes a greedy approach, promoting short paths. Traditional source-destination path planning problems and waypoint-based path planning problems can be trivially reformulated as waypoint group generalized path planning problems that our new solution in this article can handle. Our new algorithm has a competitive running time and is scalable with the number of UAS requests.

III. MOTIVATION AND PROBLEM DEFINITION

We begin this section with a brief overview of the problem domain for the generalized path planning problem. Many applications are using drone technology recently such as surveillance, aerial photography, shipping and delivery, geographic mapping, search and rescue, and weather forecast.

According to Statista [2], there were 372,000 commercial drones registered in U.S. in 2021, and this number is expected to grow in the near future. New rules and regulation are needed to establish how the airspace will be shared. One the most important and relevant research problems is the UAS path planning. Challenges associated to this problem are the physical limitations of the vehicles (mass, flight time, vehicle dynamics), sharing the airspace, safety of the vehicles, collision detection and resolution, scalability, efficiency, and seamless integration of vehicles from different commercial organizations and recreational drones.

The constituents of a UTM system are operators (companies such as UPS, Amazon, Walmart) and the UTM path planner. Operators may send planning requests to the UTM planner at any time. A request contains the start location(s), the destination location(s), a sequence of waypoint groups and time constraints for navigation between groups. A group has one or more waypoints. The computed trajectory will pass through exactly one waypoint in each group, in the given group sequence. Examples of requests are delivery of items from one of the Amazon warehouses to a client, surveillance from a start location to destination location traversing several points of interest.

Giving the UAS the option to pass through any waypoint in a group of waypoints along the path defined by such groups generalizes the path planning problem. It is suitable in scenarios where there are several options to pick up or deliver packages and for scenarios where charging stations can be used to extend flight range. Adding charger groups to a request improves considerably the request admission ratio, as seen in the performance evaluation Section V-C. The objective is to design an efficient, on-demand path planning mechanism that produces collision-free trajectories.

We assume that all UAS's have the same maximum speed v_{max} and they must be separated by a distance greater than or equal to the safety distance d when in flight in order to be collision-free.

Our proposed mechanism for path planning addresses the following aspects:

- UAS serviced by different operators
- pre-flight collision detection and resolution
- scalability with large scenarios: thousands of concurrent flights in a shared city-wide airspace.

IV. GENERALIZED PATH PLANNING MECHANISM WITH A SPACE-TIME GRAPH

In this section we propose a generalized path planning mechanism for UTM systems with pre-flight CDR. This mechanism will ensure that any new path planned will not interfere with other in-flight UAS. In prior work ([6], [18], [19]) we devised the space-time graph framework upon which we built the contributions in this article. We summarize it in Section IV-A. We continue in Section IV-B with the generalized path planning architecture, the planning algorithm, and the path validation algorithm. Section IV-C discusses issues and trade-offs from using a graph for trajectory planning. Section IV-D presents a use case for the UAS Energy-constrained Delivery Scheduling (UAS-ECDS) problem and shows a new and efficient solution using the generalized path planning algorithm.

TABLE 2. Configuration parameters with typical values.

Safety distance	d	10m
Time unit	δ	1s
Maximum vertex degree	α	5
Maximum UAS speed	v_{max}	10m/s
Number of groups	k	2 or more
Maximum group size	p	any
Battery life (max. flight duration)	H	300 - 900 time units

A. SPACE-TIME GRAPH MODEL

The model we introduced in [18] defines the aerial highway system as a directed graph $G(V, E)$, with UAS able fly only along the edges of the graph G and allowed to hover only in vertices in V. We assume that all UAS have the same maximum (or cruise) speed v_{max} and they must be separated in flight by a distance greater than or equal to the safety distance d in order for trajectories to be collision-free. Table 2 summarizes model parameters.

We make the following assumptions:

- UAS are capable of waypoint navigation with localization errors of a few cm.
- UAS move along the edges of the aerial highway traffic graph $G^{init}(V, E^{init})$. This is a weighted spatial graph, where the weight of an edge $dist(u, v)$ is the Euclidean distance from u to v. The graph must be 2D planar or with vertices at 3D positions so that non-adjacent edges are at minimum safety distance d.
- the minimum speed of each UAS is 0m/s, such as for VTOL (vertical takeoff and landing) UAS. The speed of each UAS is between 0m/s and v_{max}.
- Vehicles have a robust communication system, e.g., multi-carrier 5G cellular modems, to reduce the probability of outage.
- Vehicles report in real time their location to the UTM system (directly or indirectly via the operator) and follow the trajectory set by the operator or UTM.

To maintain a *collision-free trajectories*, the UTM system must ensure that the distance between UAS operated by the same or different operators is at least the safety distance d.

To enforce the safety distance requirement graph G^{init} is discretized to a *unit-graph* $G^{init}_u = (V_u, E^{init}_u)$ where the weight of each edge in E^{init}_u is in interval $(0, d]$. The edges in E^{init}_u result from dividing each spatial edge (u, v) of the original spatial graph G^{init} into segments of length d and plus a shorter segment at the end in case d does not divide evenly $d(u, v)$. The key constraint towards collision-free trajectories is that at most one UAS can traverse a unit edge at any time.

An edge (u, v) from the unit-graph is traversed by the UAS in time $\delta = d/v_{max}$. Time is discretized with 1 time unit equal to δ. If an *UAS* is located at the vertex u at time

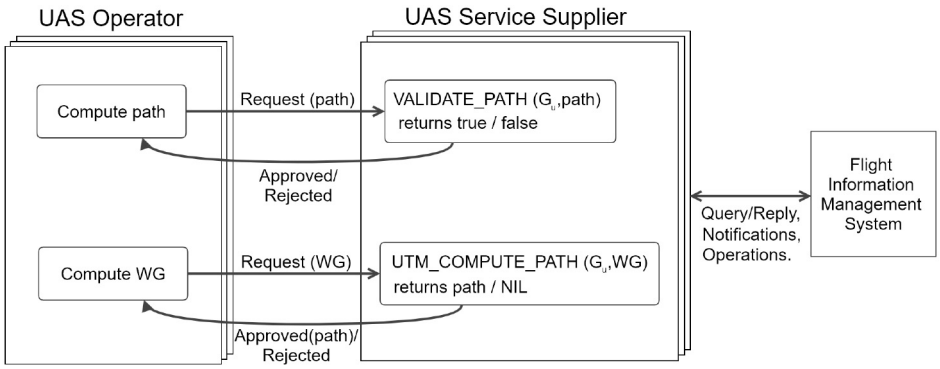

FIGURE 1. Simplified architecture that shows the communication protocol between the UAS Operator and the UAS Service Supplier.

t, then it will be in the same location at $t+1$ if the UAS pauses (e.g., hovers), or it may be located at some vertex v if the *UAS* traverses some edge (u, v).

When an UAS travels an edge (u, v) between $[t, t+1)$, we need to ensure that a safety distance d is maintained to any other UAS.

Without loss of generality, we assume in the following that $\delta = 1$. We model the availability of edges in time using a *space-time* graph denoted $G_u(t) = (V_u, E_u(t))$ where $E_u(t)$ is the set of edges which are available during the time interval $[t, t+1)$. Initially, $E_u(t) = E_u^{init}$ for each $t = 0, 1, 2, \ldots,$ however the set of edges is pruned when new UAS trajectories are scheduled.

We use the **PRUNE-RULE** algorithm from [18] to remove the edges that would conflict with a newly schedule trajectory. If a UAS traverses the edge (u, v) between $[t, t+1)$, then the pruning mechanism removes from $E_u(t)$ the edge (u, v), and all edges (a, b) with distance$\{(u, v), (a, b)\} < d$ with the following exceptions: (i) an edge (a, u) is not deleted from $E_u(t)$ if the vertices a, u, and v are collinear and $dist(a, u) = dist(u, v) = d$; (ii) an edge (v, b) is not deleted from $E_u(t)$ if the vertices u, v, and b are collinear and $dist(u, v) = dist(v, b) = d$; and (iii) the reverse edge (v, u) is not deleted from $E_u(t)$.

The edge (u, v) is removed from $E_u(t)$ since each edge in the unit-graph can be traversed by at most one UAS. The edges located at distance smaller than d are removed since they violate the safety distance d. Edges which are adjacent to (u, v) and lie on the same straight line as (u, v) will not be removed since they can be active at time $[t, t+1)$. Reverse edge are not removed since we assume they have sufficient altitude separation.

B. GENERALIZED PATH PLANNING WITH PRE-FLIGHT CDR

In this section we present the main contribution of this article: a generalized path planning mechanism with pre-flight Collision Detection and Resolution (CDR). A **client request** contains the start location (one or more nodes), the destination location (one or more nodes) and time constraints. Based on a client request, the operator prepares a **Request** to be sent to the UTM UAS Service Provider.

A simplified architecture is presented in Figure 1. Upon receiving the client request, the operator checks for feasibility and based on that prepares a Request for the UAS Service Supplier (USS) or reject the client request. Checking for feasibility include operations such as: checking that the client request can be completed by the desired time, checking that the flight time does not exceed the battery capacity limitations, etc.

The USS pre-flight CDR processes the requests in the order of arrival, although a priority mechanism could be implemented. There are two types of requests that can be initiated by the UAS Operator, see Figure 1, and either of them can be used many times interchangeable. In the first type, the UAS operator computes a path for the client request. However, the UAS operator does not have the global graph state information and the computed path may have collisions with other in-flight UAS. *Request(path)* is sent to the USS which uses the VALIDATE_PATH algorithm to determine whether the proposed path has conflicts with the ongoing traffic and returns true/false accordingly. Based on this, the proposed path is approved/rejected.

If the path is approved, then the resources are allocated by the USS using the pruning algorithm for the space-time graph. If the path is rejected, then the UAS can make another request to the USS (either type of request), or can reject the client request.

In the second request type (Figure 1), the operator formulates a generalized path request *Request(WG)* that contains a list of waypoints groups $WG = (WG_0, \ldots, WG_{n-1})$, $n \geq 1$, where WG_0 contains the possible source locations and WG_{n-1} contains the possible destination locations. Each waypoint group WG_i contains a number of nodes g_{i0}, $g_{i1}, \ldots, g_{ij}$ where $j \geq 0$ and possible time constraints for the prior waypoint groups, e.g., the flight duration between WG_k and WG_i is at most t_{ki} time units. The UAS must visit any vertex from each group, in the given group order $WG_0, WG_1, \ldots, WG_{n-1}$.

The USS uses the UTM_COMPUTE_PATH algorithm to compute a path that meet the waypoint group requirements

Algorithm 1: VALIDATE_PATH(G_u, path)

```
1: for i = 0 to path.length - 2 do
2:   if (path[i][0],path[i+1,0])∉ E_u(path[i][1]) then
3:     /* conflict detected */
4:     return false
5:   end if
6: end for
7: /* no conflict detected */
8: return true
```

Algorithm 2: UTM_COMPUTE_PATH(G_u, WG[ ][ ])

```
1: k = WG.size()
2: INIT_GROUPS(WG[ ][ ])
3: for i = 0 to k-2 do
4:   /* compute paths between group i and group i+1 */
5:   t_maxIntervalTime = COMPUTE_INTERVAL_MAX_TIME(WG[ ][ ], i, i+1)
6:   t = t_s = min{WG[i][j].wgTime for j = 0 to WG[i].size() − 1}
7:   for each vertex v at most t_maxIntervalTime hops from WG[i] do
8:     v.color = white
9:     v.π = NIL
10:    v.time = ∞
11:  end for
12:  INIT_PRIORITY_QUEUE(Q)
13:  for j = 0 to WG[i].size()-1 do
14:    WG[i][j].vertex.color = gray
15:    WG[i][j].vertex.π = NIL
16:    WG[i][j].vertex.time = WG[i][j].wgTime
17:    ENQUEUE(Q, (WG[i][j].vertex.time, WG[i][j].vertex))
18:  end for
19:  while (Q≠NIL) && (t ≤ t_s + t_maxIntervalTime) && REACHED_ALL_GROUP_VERTICES(WG[ ][ ], i+1) do
20:    (t, v) = DEQUEUE(Q)
21:    for each vertex u such that (v,u) ∈ E_u^init do
22:      if u.color ≠ white then continue
23:      if (v,u) ∈ E_u(t) then
24:        u.color = gray
25:        u.π = v
26:        u.time = t + 1
27:        if (i == k-2) && (u == WG[k-1][j].vertex for some j between 0 and WG[k-1].size()-1) && CHECK_TIME_CONSTRAINTS(WG[k-1][j]) then
28:          ASSIGN_PATH(WG[k-1][j])
29:          PRUNE-GRAPH(G_u, WG[k-1][j].path)
30:          return WG[k-1][j].path
31:        else
32:          ENQUEUE(Q, (t+1, u))
33:        end if
34:      else
35:        /* u is white and (v,u) ∉ E_u(t) */
36:        if (t+1, v) ∉ Q then ENQUEUE(Q, (t+1, v))
37:      end if
38:    end for
39:  end while
40:  for j = 0 to WG[i+1].size()-1 do
41:    if CHECK_TIME_CONSTRAINTS(WG[i+1][j]) then
42:      ASSIGN_PATH(WG[i+1][j])
43:    end if
44:  end for
45: end for
46: return NIL
```

and time constraints. The algorithm returns the first feasible path, however it can be changed to return all feasible paths. If there is no feasible path, then the algorithm returns NIL. Based on the algorithm outcome, the USS sends *Approved(path)* or *Rejected* to the UAS operator.

Next, we describe the VALIDATE_PATH algorithm. The algorithm checks whether the path is available in the space-time graph. Each path is a list of tuples. For example, a path $p = [(a, 3), (b, 4), (c, 5)]$ is a path $a \rightarrow b \rightarrow c$ where the edge (a, b) is traversed at time 3 (i.e., time interval $[3, 4)$), the edge (b, c) is traversed at time 4, and the UAS reaches the last vertex c at time 5. To check validity of p, we need to ensure the validity of the path in the space-time graph. More specifically we need to ensure that $(a, b) \in E_u(3)$ and $(b, c) \in E_u(4)$.

Lines 1 to 6 of the algorithm checks whether each edge of the *path* is in the space-time graph at the corresponding time. If at least one edge is not available, then the algorithm returns false (line 4). Otherwise, the algorithm returns true (line 8).

Next we describe the UTM_COMPUTE_PATH algorithm. The two input arguments are the space-time graph G_u and the list of waypoint groups WG. The algorithm will return a path from one of the vertices in the start waypoint group $WG[0]$ to one of the vertices in the final waypoint groups $WG[k-1]$ if such a path exist, or NIL otherwise.

We denote by k the number of waypoint groups (line 1). Line 2 calls the INIT_GROUPS algorithm, which initializes *wgTime* and *path* fields for each element of each waypoint group. $WG[i][j].wgTime$ stores the time when a path starting from an element in $WG[0]$ reaches $WG[i][j]$. The field $WG[i][j].path$ stores the path from an element in the prior waygroup $WG[i-1]$ to $WG[i][j]$. The initialization algorithm sets up the *wgTime* values to infinity and *path* to NIL since no paths are computed yet. The *wgTime* values in the initial waypoint group $WG[0]$ are set up based on the Request from the UAS Operator.

The *for* loop (lines 3 to 45) in the UTM_COMPUTE_PATH algorithm computes paths between consecutive waypoint groups $WG[i]$ and $WG[i+1]$, for $i = 0$ to $k-2$. Variable $t_{maxIntervalTime}$ is the maximum possible time for this interval and it is computed based on time restrictions included in the Request from the UAS Operator to the USS. Since each edge is traversed in 1 time unit, $t_{maxIntervalTime}$ also indicates the maximum path length from an element in $WG[i]$ to an element in $WG[i+1]$.

The pseudocode presented in lines 7 to 39 is a novel multi-source/multi-destination technique that runs on a discretized space-time unit graph. The vertices in $WG[i]$ are the source vertices, while the vertices in $WG[i+1]$ are the destination vertices.

Algorithm 3: INIT_GROUPS(WG[ ][ ])

```
1: k = WG.size()
2: for i = 0 to k-1 do
3:    for j = 0 to WG[i].size() - 1 do
4:       if i == 0 then
5:          set WG[0][j].wgTime based on the Request from the
            UAS Operator
6:       else
7:          WG[i][j].wgTime = ∞
8:       end if
9:       WG[i][j].path = NIL
10:   end for
11: end for
```

Algorithm 4: REACHED_ALL_GROUP_VERTICES(WG[ ][ ], i)

```
1: for j = 0 to WG[i].size() - 1 do
2:    if WG[i][j].wgTime == ∞ then return false
3: end for
4: return true
```

Lines 7 to 11 initialize the vertices that are at most $t_{maxIntervalTime}$ hops from the source vertices in $WG[i]$. The color is set up to white, the predecessor to NIL and the time field is set up to infinity. When a vertex is later discovered it will be colored to gray, and the predecessor and the time will be set-up accordingly.

The algorithm uses a priority queue Q. The elements of Q are tuples (t, v) where t is a time value and v is a vertex in V_u. The priority of the elements in Q is given by the first value of the tuple (i.e., by t) and elements with the same t value can be dequeued in any order.

Lines 13 to 18 initialize the source vertices from $WG[i]$ and insert them in Q. The while loop (lines 19 to 39) dequeues and processes one element at each iteration as long as the queue is not empty, the maximum duration $t_{maxIntervalTime}$ was not exceeded, and there are destination vertices which are not yet reached. For this we check the *wgTime* field of all the elements in the waypoint group $WG[i+1]$ using algorithm 4.

Let (t, v) be the current element dequeued in line 20. The *for* loop examines each neighbor u from G_u^{init}. If u is colored gray, then it has been already discovered. If u is colored white and the edge (v, u) is available at the current time (line 23), then u is discovered and the fields color, predecessor, and time are updated accordingly. If u is a final vertex, i.e., a vertex in the final waypoint group $WG[k-1]$ (line 27), then the algorithm has found a path which is returned in line 30.

If u is not a final vertex, then u is added to Q (line 32). In line 36, if u is white and the edge (v, u) is pruned, then $(t+1, v)$ is added to Q, if it is not already an element of Q. In this way the edge (v, u) is re-evaluated at the time instance $t+1$. Vertex v will be re-inserted in the queue at subsequent times as long as it has white neighbors.

When the path search for the interval $WG[i]$ to $WG[i+1]$ terminates, paths are assigned for the destination nodes in the group $WG[i+1]$, as seen on lines 40-44.

Next, we present the running time analysis. The complexity of the VALIDATE_PATH algorithm is $O(path.length)$ which can be expressed as $O(V_u)$. To analyze the UTM_COMPUTE_PATH algorithm, we denote p the maximum number of elements in a waypoint group $WG[i]$ for $i = 0, \ldots, k-1$. We also denote T the maximum interval time between two consecutive groups.

The *while* loop executes at most $O(V_u T)$ time, since each vertex is added to the queue at most once for each time value t. We denote α the maximum node out-degree. The *for* loop in line 21 adds a multiplication factor α. Therefore, the complexity for the lines 19 to 39 is $O(\alpha V_u T)$. Lines 40 to 44 assign paths to all the destination vertices in $WG[i+1]$ following the predecessor field π. The complexity for the lines 40 to 44 is $O(pV_u)$.

The overall complexity has another factor k due to the *for* loop in line 3. The PRUNE-GRAPH algorithm is run only once, when a final destination has been reached and has complexity $O(kTE_u)$. The complexity of the Algorithm 2 is therefore $O(k(\alpha V_u T + pV_u + TE_u))$.

For the performance evaluation in Section V we generate the graphs using OpenStreetMap. These graphs have a small maximum node out-degree α. For example, $\alpha = 5$ for the unit-graph of the downtown Miami map for $d = 10$, with an average degree of 1.053. Therefore, we can approximate α as constant. Also, for the generated unit-graph of the downtown Miami map $|E_u| \simeq |V_u|$. In this case, the overall running time becomes $O(k(T+p)V_u)$.

C. LIMITATIONS OF THE SPACE-TIME GRAPH APPROACH

Space-time graph planning solutions abstract over UAS capabilities and constraints. The trajectory planner computes trajectories that consist of edges in the discretized space-time unit graph. Since at most one UAS may "occupy" an edge during a time unit, a space-time edge path keeps vehicles separated by the safety distance. The graphs used for illustration in this article are directly derived from road maps on OpenStreetMap, which may not be realistic for UAS operation. The unit space graph used for simulations is a 2D and mostly planar (e.g., except for tunnels and bridges). This proof-of-concept approach for graph generation was convenient and it can easily scale up to very large maps with minimal manual work. The graph inherently avoids buildings, private property, and public places of gathering, like stadiums and parks.

The algorithms in this article will work equally well in 3D non-planar graphs, with vertices located at arbitrary heights. There is no assumption in the algorithms on its topology.

The graph construction algorithm allows one to easily create models with more realistic elements that better suit current drone dynamic/inertial constraints, such as generating aerial lanes in opposite directions properly separated, with

curved multi-segment corners, and roundabouts, as described in [25], to reduce contention in busy intersections.

We assumed an idealized model for a UAS, that is capable of precise waypoint navigation. We do not make other assumptions on the UAS performance, except for its ability to hover and its maximum speed. The discretized graph model forces vehicles to traverse each edge in a time unit δ. It does not constrain how vehicles accelerate (angular/translational) during that edge traversal. In the real world, UAS are subject to dynamic constraints (e.g., limited acceleration and rotation speeds), navigation/localization errors from operating between high-rise buildings, communication errors, and environmental factors, such as wind and precipitation. These may cause UAS to deviate from their strict space-time edge paths. What can be done?

Here are some avenues to deal with these issues. The time unit edge traversal constraint does not care about the choice of angular or linear UAS accelerations. The UAS guidance and navigation subsystems are free to compute a short term trajectory based on current conditions and vehicle capabilities. A longer time unit δ (e.g., 5 seconds instead of 1 s) may be sufficient for most UAS to traverse a longer edge while dealing with dynamic constraints and small navigation errors from wind or precipitation. The drawback of a longer time unit is a lower spatial vehicle density and higher contention for unit edges, leading up to a lower admission rate for new planning requests. Depending on the real-world application, a longer time unit may not be an issue at all and may provide a much higher safety margin.

A higher magnitude of navigation and localization error is handled by UTM by recomputing trajectories. Since UAS report periodically in real time their location, the UTM system will determine when a vehicle deviates to a degree that demands a new trajectory. The UTM planner will deallocate all space-time edges used for the remaining current trajectory and it will recompute a new trajectory starting with the UAS's current position (vertex) going to the original destination. The operator will then forward the new route to the UAS. If an unforeseen weather event forces multiple vehicles to deviate from their current plan, the UTM planner will force all affected vehicles to switch to a detached hover state that minimizes interference with other ongoing flights while it recomputes new trajectories starting from the current vertices. If a new trajectory is not possible any more, the UTM will route the UAS to a new destination (e.g., depot) in collaboration with the operator.

In case of a brief interruption of the communication channel between a UAS and the operator (and UTM), the UAS may continue its planned trajectory and rely on its onboard collision avoidance to prevent collisions for the short term. If the interruption exceeds a duration threshold the UAS may switch to a safely hover state until communication with UTM or the operator resumes. The UTM will then have to recompute trajectories for affected UAS. Specific policies will have to be formulated by regulatory agencies (e.g., FAA in the USA) that standardize UAS behaviors in case of such large-scale disruptions.

Space-time graph planning requires further investigation and refinement to fully support operational scenarios like those mentioned above. Future work will look at dealing with navigation/communication errors and dynamic vehicle rerouting. The field is in rapid evolution with directions set by regulatory bodies.

D. USE CASE: UAS ENERGY-CONSTRAINED DELIVERY SCHEDULING

In this section we present a novel solution for the UAS Energy-constrained Delivery Scheduling (UAS-ECDS) problem we introduced in [6]. The UAS-ECDS problem is about an operator finding a UAS trajectory for package delivery to a client when the operator has multiple options for start and finish warehouses and the UAS can recharge/replace its battery at multiple charging stations. As drone battery technology evolves this type of scenario will become more likely to be used in practice. This algorithm relies on the UTM system using the generalized path planner presented in Section IV-B.

In the **UAS-ECDS problem** we are given an aerial highway traffic system modeled as a directed graph $G(V, E)$. The set of vertices $C = \{c_1, \ldots, c_k\}$, $C \subset V$, is the set of charging stations. The operator formulates a request for the UTM system with the following fields: client location (a vertex in V), the set of start warehouses $W_s \subset V$, the set of final warehouses $W_f \subset V$, where $W_s \subseteq W_f$, the request start time, the deadline to reach the client, and the deadline to reach a final warehouse. The flight time is limited by the battery capacity, but the UAS operational range can be extended by stopping at charging stations to recharge or replace the battery. Our solution has the operator formulate the initial planning problem to a sequence of waypoint-group path planning requests sent to the generalized path planner used by the UTM system.

We assume that the UAS can charge at most once on the path from the start warehouse (a vertex in W_s) to the client, and at most once on the path from the client to the final warehouse (a vertex in W_f). A delivery path looks as follows: start warehouse $\rightarrow$ charging station (optional) $\rightarrow$ client $\rightarrow$ charging station (optional) $\rightarrow$ final warehouse. Charging or replacing a UAS battery takes a time that is a configuration parameter. Charging takes a relatively small time if the only operation performed is to replace the battery. There are few ways in which we can model the battery constraint UAS delivery. Since in our framework UAS travel along unit-edges and each UAS takes δ time to traverse an edge, we assume that a fully charged battery lasts $H \times \delta$ time, thus the UAS can traverse at most H unit edges before recharging.

A request has the following fields: $req = (clientVertex, W_s, W_f, C, t_s, maxTimeClient, maxTimeWf)$, where $clientVertex$ is the client location, W_s is the set of start warehouses, W_f is the set of final warehouses, C is the set of charging stations, t_s is the request start time, $maxTimeClient$ is the deadline to

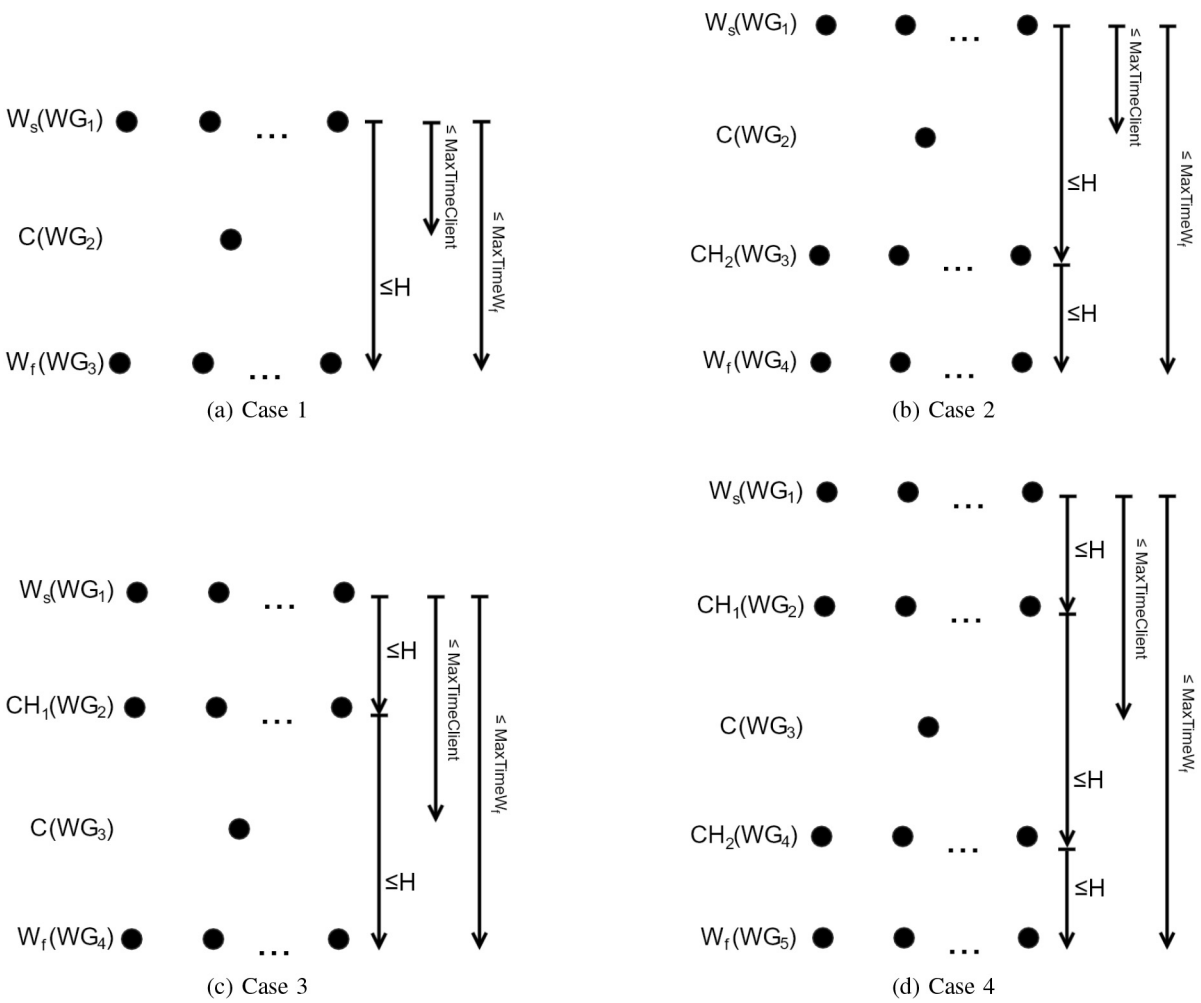

FIGURE 2. Four cases for computing a path for the UAS-ECDS problem.

reach the client, and *maxTimeWf* is the deadline to reach a final warehouse.

When a UAS operator needs a new package delivery trajectory it considers four possible cases, shown in Figure 2, in the order listed below. For each of the four cases, the operator creates an admission *Request* for a waypoint group path that also includes corresponding time constraints and then it sends it to the UTM planner. The UTM planner executes the generalized path planning algorithm and replies with the planning result: a message with *success* code and the trajectory, or a message with *reject* code. In case of success, the operator concludes the admission process and it will send the returned trajectory to the UAS before the scheduled take-off time. In case of rejection the operator will attempt the next case from the four. If all four cases fail, the operator gives up, as no feasible trajectory was found by the UTM system.

Initially, the operator will attempt a waypoint group path without using any charging station. Each successive attempted case uses progressively more charging station groups. This is to find shorter paths, rather than unnecessary long ones involving charging stations. Since the generalized path planning algorithm is greedy, this iterative solution also follows a greedy strategy.

We discuss next the four cases attempted by the operator in sequence, also shown in Figure 2. The operator models the sets of start warehouses and finish warehouses as the start ($W_s(WG_1)$) and end ($W_f(WG_x)$) waypoint groups. A group C is defined for the client vertex. Additional groups represent the charging stations, CH_1 and CH_2. Each subfigure illustrates the groups from the admission request from top to bottom. A group (e.g., WG_1) has some number of vertices (horizontal line of black dots). On the right-hand side the subfigures show the time constraints to reach waypoints from the request waypoint groups.

- **Case 1: no charging stations**. Paths (see Figure 2(a)) are in the form: w $\rightarrow$ client $\rightarrow$ wf, where w is a starting warehouse $w \in W_s$, and wf is a final warehouse $\in W_f$. There are 3 waypoint groups WG defined as follows: $WG_0 = W_s$, $WG_1 = \{ClientVertex\}$, and $WG_2 = W_f$. The time constrains are defined as follows: (i) $Time(WG_0, WG_2) \leq H\delta$ ensures that the

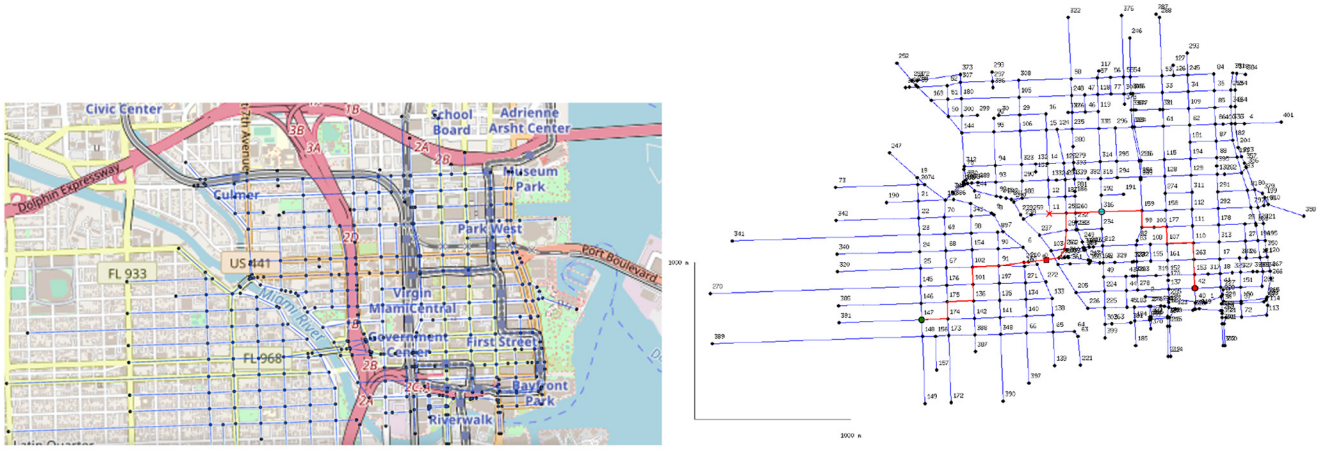

FIGURE 3. (a) Miami map used for evaluation. (b) Sample route for the Miami graph. The green circle shows the start warehouse, the red X marks the client, the red circle is the finish warehouse and the light blue circle is a charging station.

battery resources are sufficient for the whole path, (ii) $Time(WG_0, WG_1) \leq MaxTimeClient$ ensures that the UAS reaches the client before the deadline, and (iii) $Time(WG_0, WG_2) \leq MaxTimeW_f$ means that the UAS meets the deadline for reaching a final warehouse. This case is attempted only if the minimum total distance between a source warehouse, the client, and a destination warehouse is shorter than the maximum flight distance ($H\delta v_{max}$).

- **Case 2: charging stations after client**. Paths (see Figure 2(b)) are in the form: w $\rightarrow$ client $\rightarrow$ ch2 $\rightarrow$ wf, where ch2 is a charging station $ch2 \in C$. There are 4 waypoint groups WG defined as follows: $WG_0 = W_s$, $WG_1 = \{ClientVertex\}$, $WG_2 = C$ where C is the set of charging stations, and $WG_3 = W_f$. The time constrains are defined as follows: (i) $Time(WG_0, WG_2) \leq H\delta$, (ii) $Time(WG_2, WG_3) \leq H\delta$, (iii) $Time(WG_0, WG_1) \leq MaxTimeClient$, and (iv) $Time(WG_0, WG_3) \leq MaxTimeW_f$. The first two time constraints ensure that the battery resource constraints are met from the start warehouse to the charging station and from the charging station to the final warehouse. The last two constraints ensure that the time constraints to the client and to the final warehouse are met.
- **Case 3: charging stations before client**. Paths (see Figure 2(c)) are in the form: w $\rightarrow$ ch1 $\rightarrow$ client $\rightarrow$ wf, where ch1 is a charging station $ch1 \in C$. There are 4 waypoint groups defined as follows: $WG_0 = W_s$, $WG_1 = C$, $WG_2 = \{ClientVertex\}$, and $WG_3 = W_f$. The time constrains are defined as follows: (i) $Time(WG_0, WG_1) \leq H\delta$, (ii) $Time(WG_1, WG_3) \leq H\delta$, (iii) $Time(WG_0, WG_2) \leq MaxTimeClient$, and (iv) $Time(WG_0, WG_3) \leq MaxTimeW_f$. Their description is similar to Case 2.
- **Case 4: charging stations before and after client**. Paths (see Figure 2(d)) are in the form: w $\rightarrow$ ch1 $\rightarrow$ client $\rightarrow$ ch2 $\rightarrow$ wf. There are 5 waypoint groups defined as follows: $WG_0 = W_s$, $WG_1 = C$, $WG_2 = \{ClientVertex\}$, $WG_3 = C$ and $WG_4 = W_f$. The time constrains are defined as follows: (i) $Time(WG_0, WG_1) \leq H\delta$, (ii) $Time(WG_1, WG_3) \leq H\delta$, (iii) $Time(WG_3, WG_4) \leq H\delta$, (iv) $Time(WG_0, WG_2) \leq MaxTimeClient$, and (v) $Time(WG_0, WG_4) \leq MaxTimeW_f$. The first three constraints ensure the battery resource constraints are met from the start warehouse to the first charging station, between the two charging stations, and from the second charging station to the final warehouse. The last two time constrains are similar to those in the prior cases.

Section V evaluates the performance of the proposed mechanism compared to the algorithm from [6].

V. PERFORMANCE EVALUATION

In this section we analyze the performance of the UTM_COMPUTE_PATH algorithm for generalized path planning and that of the UAS-ECDS solution described in Section IV-D. We compare the latter with the original delivery algorithm from [6].

A. SIMULATION ENVIRONMENT

We implemented the algorithms from the previous section and a simulation framework in Python 3 [20]. The simulations ran on the Florida Atlantic University high performance computer cluster [21], [22]. Program execution was handled by SLURM [23], the scalable cluster management and job scheduling system for Linux clusters. The computers that ran the simulations are Dell PowerEdge C4140/013M88 with 32 Intel Xeon Gold 6130 CPUs @ 2.10 GHz, 187 GB memory, and NVidia GPUs.

We used the downtown Miami map (FL USA) from [6], see Figure 3. The map was extracted from the Open Street Maps website [24]. The Miami (3700 $\times$ 2400 m) unit graph has 6946 vertices and 14264 edges. Figure 3(a) shows the original directed graph representation $G(V, E)$ over-imposed

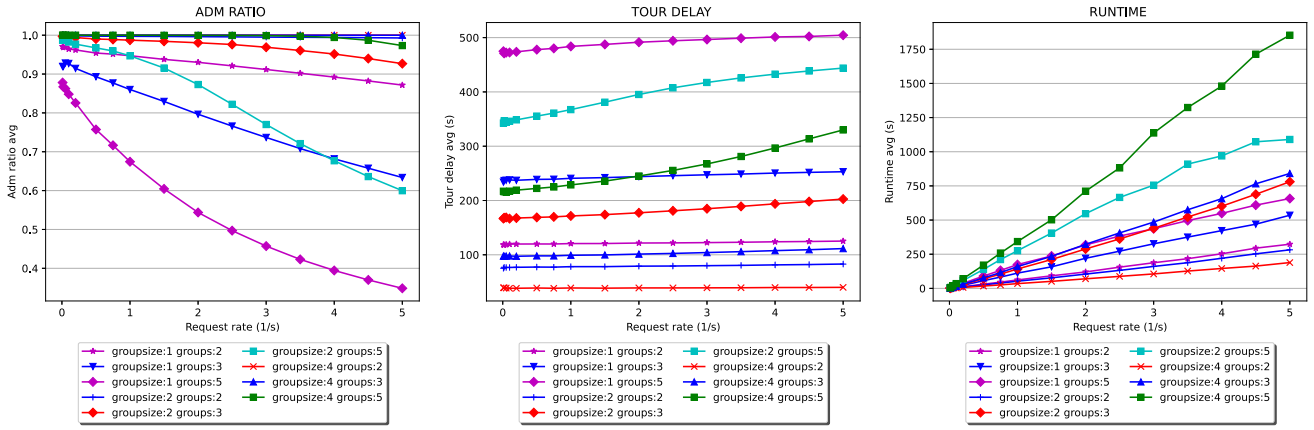

FIGURE 4. Experiment 1 focuses on request rate and presents results for admission ratio, tour delay and running time.

TABLE 3. Simulation parameters common to both performance evaluation studies.

Miami Map	Approx. 3.7km x 2.4km
Miami Graphs	$\|V\| = 402, \|E\| = 590$ Unit graph: $\|V_u\| = 6946, \|E_u\| = 14264$
Safety Distance d	10m
Time Unit δ	1s
Max. Speed v_{max}	10m/s

TABLE 4. Simulation parameters used for UTM_COMPUTE_PATH experiments.

Safety Distance d	10m
Time Unit δ	1s
Max. Speed v_{max}	10m/s
Total Simulation Time	4000s
Number of Iterations	10
UAS Range	32 - 19000
Request Rate Range	0.01/s - 5/s
Max. Flight Time or Battery Life	200s
Max. Route Duration	*Max. Flight Time* $\times(group\# - 1)$

on the geographic map. Table 3 lists simulation parameters common to both studies. Our algorithms rely on the grid system from [18] for caching edge neighborhood information to speed up edge pruning when updating the space-time edge availability after a successful admission.

Both performance analyses have two common performance metrics. The *runtime* measurement represents how fast an algorithm performs a planning algorithm for all admission requests sent during the entire simulation. It depends on the admission request count, which is equal to the product of the *request rate* simulation variable and the simulation time during which new requests are submitted. The *admission ratio* measures how many requests were successfully admitted divided by the total number of requests submitted for admission. Other performance metrics are introduced later for specific experiments.

B. THE WAYPOINT GROUP TOUR PLANNER EVALUATION

The UTM_COMPUTE_PATH algorithm was evaluated with three experiments, each with a different range of variables and using the common configuration parameters listed in Table 4.

The experiment variables fixed for a single simulation scenario are the request rate, the number of groups, and the number of waypoints per group, which is the same for all groups in a request. The maximum allowed route duration for each request is limited by the number of groups and the maximum flight time allowed between two waypoints (200s), and it varies between 200s (2 groups) and 1400s (8 groups).

Each simulation scenario involves 10 random runs (iterations) with identical variables but different requests, with waypoint vertices selected at random in each group so there is at least one feasible path with the constrained total duration in an idle graph from the first to the last group. Thus, ignoring any load on the graph resources, the maximal admission ratio would be 100%. The admission ratio is always less than 100% for cases with heavy traffic.

The performance metrics for this study are the algorithm CPU runtime, the average tour delay, and the average admission ratio. The measurements for each scenario are averaged and the mean is reported. The variables for the three experiments are listed in Table 5.

In the first experiment we look at the impact of the request rate on performance. We vary the request admission rate between 0.01/s (corresponding to N = 32-38 UAS requests overall) to 5/s (N = 16000-19000). The group count changes from 2 to 5, and the waypoint count per group is 1, 2, or 4. Different values for N are obtained for each combination of group count/group size.

The performance results are shown in Figure 4. The first chart shows the average admission ratio. We expect that a higher load on space-time graph resources that can't be shared (space-time edges) will lower the admission ratio. When space-time graph edges connecting two different vertices are busy the planner will compute a path with higher cost in space-time. Longer space-time paths will have a higher probability of hitting the maximum flight duration scenario parameter limit, leading to a lower admission ratio.

TABLE 5. Variable values used for the three UTM_COMPUTE_PATH experiments.

Experiment#	Request Rate $[s^{-1}]$	Group Count	Waypoints / group
1	0.01 0.02 0.05 0.10 0.2 0.5 0.75 1 1.5 2 2.5 3 3.5 4 4.5 5	2, 3, 4, 5	1, 2, 4
2	0.01 0.1 1 2 4 5	2, 4, 6	1 - 10
3	1, 5	2 - 8	1, 2, 4, 8

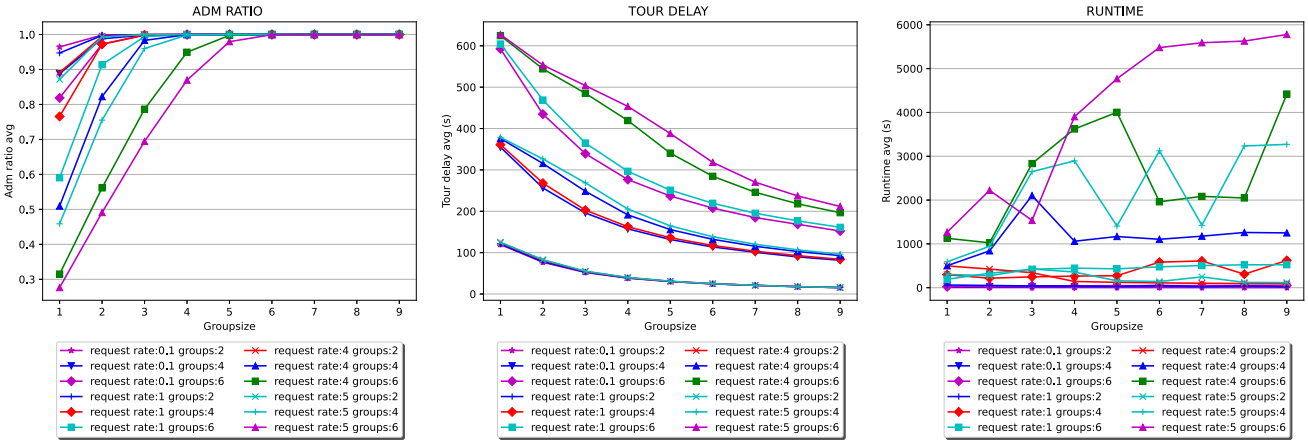

FIGURE 5. Experiment 2 focuses on group size and presents results for admission ratio, tour delay and running time.

The chart shows that for a higher request rate the admission ratio drops, as expected, because of the increased contention for resources. We also notice that for the same group size the admission ratio will decrease significantly with more waypoint groups for a path. This is because contention for graph resources is proportional with the path length, which is on average proportional with the number of groups. The admission ratio drop is more dramatic for small group sizes.

The second chart of the Figure 4 shows the average tour delay. As expected, with constant group size the tour delay is proportional with $groups - 1$, since each additional leg of a path contributes on average a similar delay between waypoints. Noticeable is the very slow growth of the tour delay with higher request rate and a low group count of 2. The load increase on graph resources up to 5 requests/s is not enough for a sharper delay growth. That can be observed when the group number goes up to 5. The extra load (e.g., occupancy) on graph resources from more UAS enroute for 5 groups is 4 times that of of scenarios with 2 groups, causing a clear growth trend. With $groups = 5$ and $groupsize = 4$ the tour delay growth trend is accelerating when the request rate is increasing. Another important observation is that adding more waypoints to the group size reduces the tour delay significantly: between 12-27% for $groups = 5$ and going from 1 waypoint to 2 per group, for a request rate of 5/s. The drop is more dramatic when going from $groupsize = 1$ to 5 per group: 34%-54% drop in tour delay. The benefit from having more options is clear.

The third chart in Figure 4 displays the runtime. As first derived in Section IV-B, the theoretical complexity of Algorithm 2 is $O(k(\alpha V_u T + pV_u + TE_u))$, and this is just for one request. Here, k is the number of waypoint groups, T is the maximum interval time between two consecutive groups, p is the maximum group size. The runtime should be directly proportional to $p \cdot k$, the product of the maximal group size and the group count. The runtime measured includes all requests during a scenario, so it depends linearly on the request rate, as expected. However, the runtime measurement does not seem to be proportional to $p \cdot k$. The reason for that is that the theoretical formula is an upper bound and it does not consider the effects of dwindling resource (edges and vertices) availability with higher request rates.

The second experiment varies the group size from 1 to 9, with charts in Figure 5. The first chart shows the admission ratio for various request rate - group count combinations. The request rate values are 0.1/s (N = 300 − 380), 1/s (N = 3000 − 3800), 4/s (N = 12000 − 15200), and 5/s (N = 15000 − 19000). The number of groups is 2, 4, and 6. As expected, the admission ratio grows quickly when the group size increases from 1 to 9. A group size of 6 brings it to or very close to 100% for all parameter combinations. A higher waypoint group size increases the probability of admission - up to 100%, even in busy scenarios with 19000 requests (request rate = 5/s). The admission ratio is reduced by heavier contention for graph resources leading up to longer paths that have a higher probability of missing the maximum allowed delay per inter-waypoint path – the maximum flight time. We see that for scenarios with 4 groups and 6 groups that start their growth trend from much lower admission ratios (e.g., 27% for request rate = 5 and groups = 6).

The second chart from Figure 5 illustrates the dependency of the tour delay on the group size, with group counts (2, 4, 6) and request rates (0.1, 1, 4, and 5). With more waypoints per group, paths are shorter due to more options and there

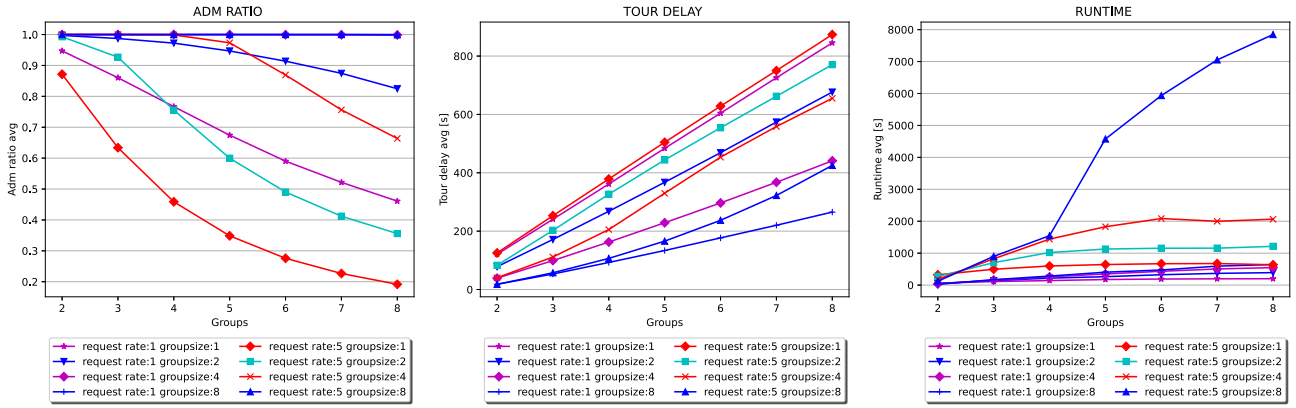

FIGURE 6. Experiment 3 focuses on the number of waypoint groups and presents results for admission ratio, tour delay and running time.

is also a higher probability of admission. The drop in the tour delay is more pronounced for paths with more groups.

The runtime is shown in the third chart in Figure 5. We notice a higher variation for scenarios with higher request rates (4/s, 5/s) and more groups (4 and 6). It seems there is no consistent dependency on group size across the board, considering all other variables constant. In theory, the upper bound for runtime should be proportional to the group size. In each iteration, the algorithm tries to find a path connecting reachable waypoints from the current group to all waypoints in the next group. In practice, it is also determined by resource availability. We notice that for a request rate of 5/s and 6 groups there is a clear growth trend, since the graph is quite busy.

The charts in Figure 6 show the results from the third experiment, with the number of groups growing from 2 to 8. The request rate is one of 1 and 5/s, and the group size ranges in {1, 2, 4, 8}.

The first chart shows the admission ratio. A higher waypoint group number means longer paths and higher edge occupancy due to the proportionally higher number of UAS being concurrently enroute. This causes a clear drop in admission ratio with more groups. The drop is sharper with small group sizes, like 1 and 2, since the probability of finding a path with fewer waypoint options is lower. This is very visible for request rate 5/s. The admission rate falls from 87% with group size 1 to 19%, with 8 groups. Increasing the group size from 1 to 4 raises the admission ratio for 8 groups to 66%, and to 99.91% for a group size of 8 waypoints.

The average tour delay is illustrated on the second chart in Figure 6. The linear dependency of the tour delay on the group count is clear. For higher request rate (5/s) and bigger groups (4 and 8) the tour delay has a slight upswing with more groups because of the increased contention for space-time graph resources.

The third chart in Figure 6 shows the runtime dependency on the group count. The upper bound for the algorithm complexity is linear in the group count (variable k in its pseudo-code). This is apparent for small values of the group count. The downward swing of the runtime for higher group counts and low group sizes is correlated with a drop in the admission ratio in this cases, as discussed above. Basically, the path search algorithm fails faster for an increasing number of admission requests.

TABLE 6. Simulation parameters used for the delivery problem comparison study.

Safety Distance d	10m
Time Unit δ	1s
Max. Speed v_{max}	10m/s
Total Simulation Time	4000s
Number of Iterations	5
Max. Flight Time or Battery Life	250s
Max. Return Time to Warehouse	1000s
Max. Route Duration to Client	250s
Request Rate Range	0.01/s - 5/s
Number of Warehouses "Nw"	1 - 9
Number of Chargers	0 - 8

C. COMPARISON OF THE PACKAGE DELIVERY ALGORITHM (SECTION IV-D) WITH THE ALGORITHM FROM [6]

Next, we compare the performance of the package delivery algorithm from Section IV-D and abbreviated in charts with letter "w") with the original trajectory planner from related work [6], abbreviated in the charts with letter "d".

We used the same simulation environment described at the top of this section for both algorithms, including the same downtown Miami map and graph parameters from Table 3. The scenario variables and configuration parameters are summarized in Table 6.

The maximum flight time represents the battery lifetime constraint. A UAS must reach a charger or a finish group warehouse within this time interval. The maximum route duration to client is the maximum allowed time to reach a client from a starting warehouse. The performance metrics are the algorithm runtime, the time to reach the client vertex, and the request admission ratio, defined as in Section V-B.

The independent variables in this study are: the request rate, the number of charging stations ("Nch"), and the number of warehouses ("Nw"), equal for both the start and finish warehouse groups.

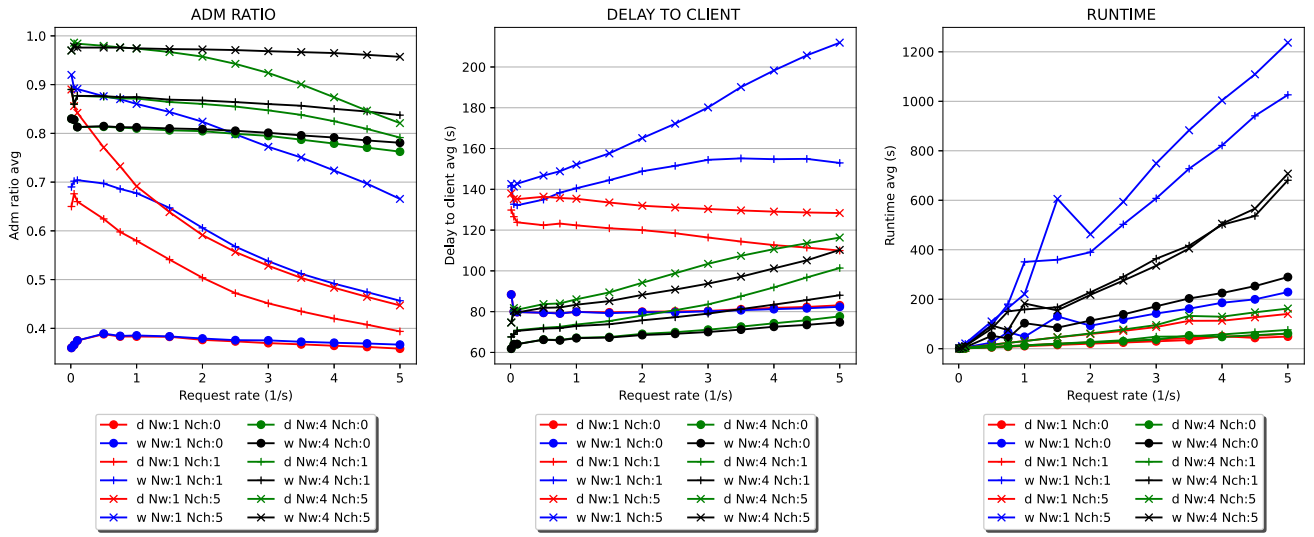

FIGURE 7. Performance results for the comparison study with request rate variation.

The charger vertices are selected from unit graph vertices nearest to the $k-means$ of all unit graph vertices in order to improve their even spatial distribution. The k parameter for the $k-means$ clustering algorithm is equal to the number of chargers (Nch) for a particular scenario.

The first chart in Figure 7 shows the admission ratio for both algorithms, where "d" stands for the original delivery algorithms and "w" for the waypoint group-based delivery algorithms. The request rate grows from 0.01/s (N = 20) to 5/s (N = 10000) and the number of start/finish warehouses is 1 or 4 and the number of chargers is 0, 1, or 5. Not surprising, with a higher request rate the admission ratio trends down from the increased contention that causes an increasing number of requests to miss deadlines. We notice that for request rates exceeding 0.5/s and for all warehouse-charger count combinations, the admission ratio is significantly better for the waypoint group-based delivery algorithm, and the advantage just keeps growing with higher request rates. The path planning algorithm proposed in this article does better by searching paths directly in the space-time graph, as opposed to the original delivery algorithm that finds shortest paths in the spatial domain (space-only graph) and then deals with resource contention by scheduling time-only edges, effectively delaying UAS progress by making them wait, instead of finding alternate space-time routes.

The second important observation is that addition of well-placed charging stations raises the admission ratio dramatically for low request rates. For high request rates it is much more effective to have several warehouses, instead of one.

The second chart in Figure 7 measures the delay to client metric, i.e., the time to reach the client from any of the start warehouses, when the request rate varies from 0.01/s to 5/s, for various combinations of Nw and Nch. For scenarios with one warehouse per groups (Nw = 1) the new waypoint group-based delivery algorithm does not perform better than the old algorithm. However, with 4 warehouses the new algorithm achieves a smaller client delay.

The third chart in Figure 7 compares the runtime of the two algorithms when the request rate varies in the 0.01/s - 5/s range, and for different combinations for Nw and Nch. In general, the runtime is proportional with the request rate, with some variation between 0.5/s and 1/s for the new waypoint group based algorithm. The original delivery algorithm is consistently much faster, as it operates mainly in a 2D search space (the space graph), and not in a space-time graph. The trade-off for a higher execution time, however, is much better admission ratio and higher graph resource utilization, i.e., a higher UAS traffic density.

In a second experiment we vary the warehouse count per group, keep the request rate at 5/s and change the number of chargers from 0, to 3, to 6. Figure 8 shows the simulation results.

The first chart displays the admission ratio. As the number of warehouse grows, the new waypoint group delivery algorithm outperforms consistently the original delivery algorithm, by 20% for low Nw. Adding more chargers to a request further increases the admission ratio, but more so for the new delivery algorithm.

The second chart in Figure 8 displays the path delay to client metric under the same set of variables. We notice that the new waypoint group based delivery algorithm starts outperforming the original delivery algorithm when the number of warehouses reaches 4, for scenarios with 3 or more chargers. It always gives lower delays for scenarios with NCh = 0. The new algorithm proves to be more effective, with lower delay metrics and better admission ratios under higher resource contention and with more options for warehouses and chargers.

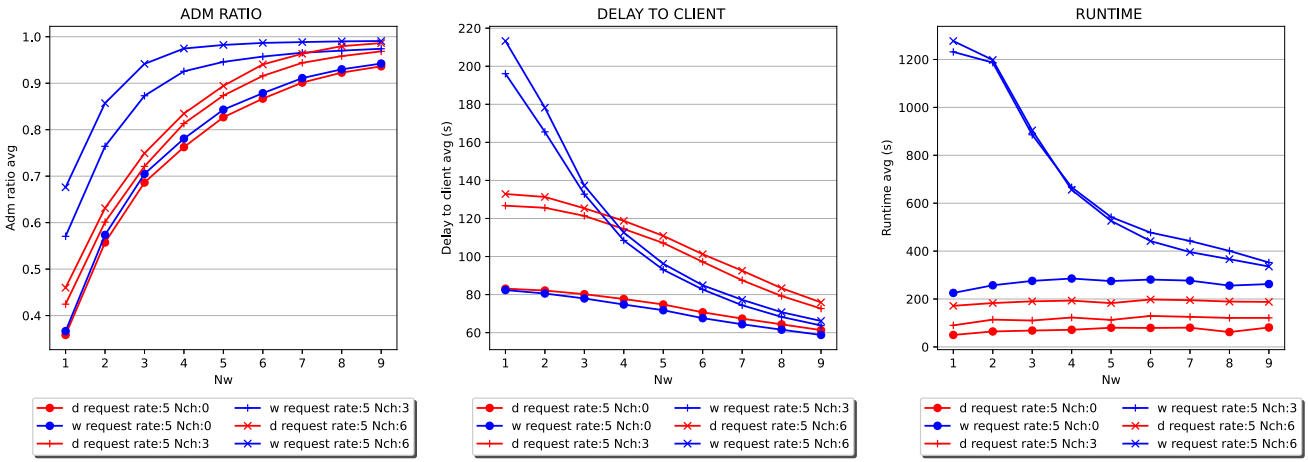

FIGURE 8. Performance results for the comparison study when the number of warehouses per group changes from 1 to 9.

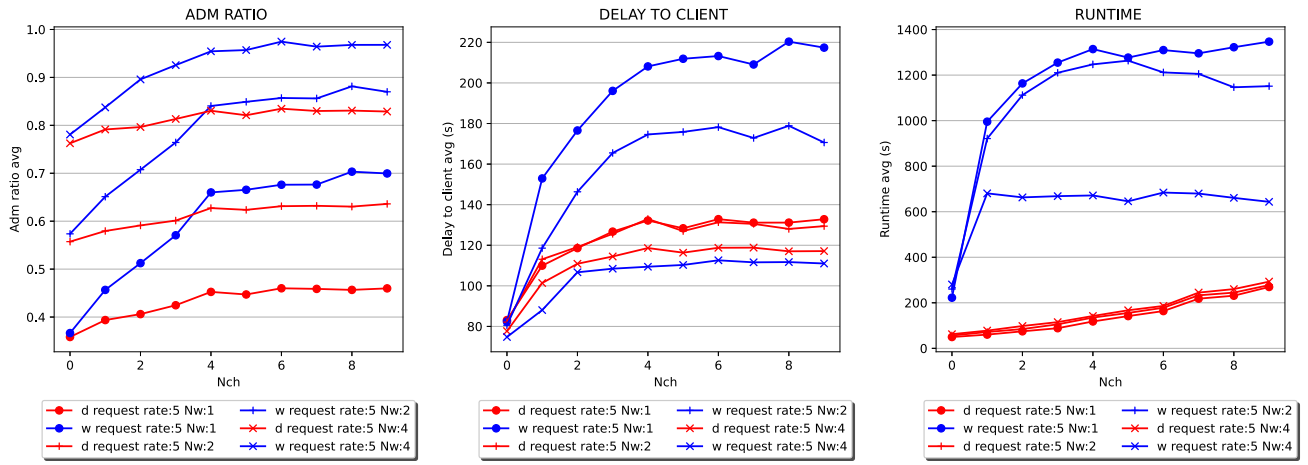

FIGURE 9. Performance results for the comparison study when the number of chargers per group changes from 0 to 9 and the request rate is 5/s.

The third chart in Figure 8 shows the algorithm runtime. For all scenarios the original algorithm has a quicker execution time, regardless of Nw. With a growing Nw, the runtime for the new algorithm drops and gets just 70% higher than that of the original delivery algorithm for request rate at 0.5/s and Nch = 6. The new algorithm achieves consistently a higher admission ratio than the old algorithm, and a much higher one for a low number of warehouses, Nw. This translates to a longer algorithm runtime since the search in the space-time graph does not fail so soon.

The third experiment looks at the impact of the number of chargers on performance metrics when the request rate is 5/s and Nw is 1, 2, or 4. Results are shown in Figure 9. The admission ratio is shown on the first chart. The first observation is that the new algorithm always provides higher admission ratios compared to the original algorithm. Also, the growth is accelerated mostly up to Nch = 4, after which the growth is slower. We conclude that the new algorithm benefits from adding charging stations, resulting in an improvement in admission ratio and better use of the UAS airspace.

The second chart in the Figure 9 shows the delay to client metric. With Nch $>$ 0 the delay is longer for the new algorithm, except for the Nw = 4 case, that consistently outperforms the original algorithm. This chart shows us that for this configuration, adding more than 4 chargers does not improve the delay metric, although it gives slightly better admission ratios.

The algorithm runtime is displayed in the third chart of the Figure 9. The new algorithm has much longer execution time compared to the original algorithm, as seen already in the prior runtime charts.

D. SUMMARY

The results of the UTM_COMPUTE_PATH algorithm simulations from Figures 4-6 showed that:

- a higher request rate causes increased contention on space-time graph edges, causing the planner to compute longer paths leading to a drop to admission ratio, and a linear increase in the algorithm runtime;
- adding more waypoints to each group

  - quickly increases the admission ratio for all scenario configurations. With 6 vertices in each group the admission ratio reached very close to 100% even for scenarios with high request rates (5/s, N = 19000) and 6 groups.
  - the tour delay drops gradually with a slowing trend;
- adding more waypoint groups to a path:
  - reduces the admission rate because of increased contention for space-time graph resources;
  - increases the tour delay linearly;
  - generally also increases the runtime linearly, but with a sublinear growth trend for higher request rates because of an increasing number of admissions failing that cut short the algorithm.

The comparison study of the UAS-ECDS algorithm from Section IV-D with the original delivery algorithm from [6] demonstrated the following:

- increasing the request rate:
  - the admission ratio trends down from the increased contention for graph resources (edges). The new waypoint group-based delivery algorithm does considerably better for request rates exceeding 0.5/s, for all warehouse-charger count combinations.
  - the delay to client metric is lower for the new algorithm in cases with more than one warehouses per start/finish group
  - the runtime grows linearly and is consistently higher for the new algorithm.
- increasing the number of warehouses in the start and finish groups:
  - the new algorithm has consistently higher admission ratio (up to a 20% difference);
  - the new algorithm starts to generate paths with lower delay to client when the number of warehouses reaches 4, for scenarios with 3 or more chargers;
  - the original algorithm has a lower runtime;
- increasing the number of number of charging stations:
  - the new delivery algorithm always provide higher admission ratios compared to the original algorithm, with the growing trend slowing after reaching Nch = 4;
  - the new algorithm gets a lower delay to client only for scenarios with 4 warehouses per start/finish group
  - the new algorithm has consistently higher runtime then the original one.

We note that adding a few well-placed charging stations to a request raises the admission ratio dramatically for low request rates. For high request rates it is much more effective to add multiple warehouses to a request.

We conclude the performance evaluation summary by stating that the new waypoint group-based delivery solution outperforms the original one from [6] for admission ratio in most scenarios, for delay to client in scenarios with higher number of warehouses, and trades off these qualities for a higher runtime. In practical UTM implementations the planner algorithm would run on a powerful computer cluster with extensive parallelization, so that a higher runtime can be mitigated.

VI. CONCLUSION

This paper proposes a novel mechanism for generalized path planning using a space-time graph framework and a path validation algorithm. A generalized path is defined by a sequence of waypoint groups. The UAS must pass through exactly one waypoint in each of the groups and the computed trajectory must be collision-free, keeping all UAS with a minimum safety distance separation. The space-time graph discretization together with space-time edge pruning for allocated edges ensure all trajectories are collision-free by construction. To compute the space-time trajectory, we devised a novel multi-source/multi-destination graph search algorithm that runs on the space-time graph. The path validation algorithm verifies that all path space time edges are available in the space-time graph. We use our generalized path planner to solve the UAS-ECDS problem for package delivery with multiple warehouses and multiple charging stations. Simulation results show that our path planning algorithm is efficient asymptotically and it is scalable with the number of requests and graph size.

REFERENCES

[1] "Business insider, drone market outlook in 2021: Industry growth trends, market stats and forecast." Nov. 2021. [Online]. Available: https://www.businessinsider.com/drone-industry-analysis-market-trends-growth-forecasts

[2] Statista. "Drone market revenue worldwide from 2019 to 2025." Nov. 2021. [Online]. Available: https://www.statista.com/statistics/1200348/drone-market-revenue-worldwide/

[3] The Federal Aviation Administration (FAA). "Aerospace forecast fiscal years (FY) 2020–2040." Nov. 2021. [Online]. Available: https://www.faa.gov/newsroom/federal-aviation-administration-faa-aerospace-forecast-fiscal-years-fy-2020-2040

[4] "The Federal Aviation Administration (FAA), unmanned aircraft systems (UAS) traffic management (UTM) concept of operations version 2.0, Washington, DC, USA." Nov. 2021. [Online]. Available: https://www.faa.gov/uas/research_development/traffic_management/

[5] F. Ho, R. Geraldes, A. Gonçalves, M. Cavazza, and H. Prendinger, "Improved conflict detection and resolution for service UAVs in shared airspace," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1231–1242, Feb. 2019, doi: 10.1109/TVT.2018.2889459.

[6] R. Papa, I. Cardei, and M. Cardei, "Energy-constrained drone delivery scheduling," in *Proc. Int. Conf. Comb. Optim. Appl. (COCOA)*, Dec. 2020, pp. 125–139.

[7] S. H. James and R. N. Raheb, "Path planning for critical ATM/UTM areas," in *Proc. IEEE/AIAA 38th Digit. Avionics Syst. Conf. (DASC)*, 2019, pp. 1–6, doi: 10.1109/DASC43569.2019.9081662.

[8] S. James, R. Raheb, and A. Hudak, "UAV swarm path planning," in *Proc. Integr. Commun. Navig. Surveillance Conf. (ICNS)*, 2020, pp. 2G3-1–2G3-12, doi: 10.1109/ICNS50378.2020.9223005.

[9] V. Jeauneau, L. Jouanneau, and A. Kotenkoff, "Path planner methods for UAVs in real environment," *IFAC-PapersOnLine*, vol. 51, no. 22, pp. 292–297, 2018, doi: 10.1016/j.ifacol.2018.11.557.

[10] A. Chakrabarty, V. Stepanyan, K. Krishnakumar, and C. Ippolito, "Real-time path planning for multi-copters flying in UTM-TCL4," presented at the AIAA Scitech Forum, San Diego, CA, USA, 2019, pp. 1–16.
[11] M. Egorov, V. Kuroda, and P. Sachs, "Encounter aware flight planning in the unmanned airspace," in *Proc. Integr. Commun. Navig. Surveillance Conf. (ICNS)*, 2019, pp. 1–7, doi: 10.1109/ICNSURV.2019.8735399.
[12] J. T. Betts, "Survey of numerical methods for trajectory optimization," *J. Guid. Control Dyn.*, vol. 21, no. 2, pp. 193–207, 1998.
[13] D. Sacharny, T. Henderson, and M. Cline, "An efficient strategic deconfliction algorithm for large-scale UAS traffic management," School Comput., Univ. Utah, Salt Lake City, UT, USA, Rep. UUCS-20-010, 2020.
[14] National Academies of Sciences, Engineering, and Medicine, *Roundabouts: An Informational Guide*, 2nd ed. Washington, DC, USA: Nat. Acad. Press, 2010. [Online]. Available: https://doi.org/10.17226/22914
[15] M. Stevens and E. Atkins, "Geofence definition and deconfliction for UAS traffic management," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 9, pp. 5880–5889, Sep. 2021, doi: 10.1109/TITS.2020.3040595.
[16] D. Sacharny, T. Henderson, and E. Guo, "A DDDAS protocol for real-time large-scale UAS flight coordination," in *Proc. ACM DDDAS*, 2020, pp. 49–56, doi: 10.1007/978-3-030-61725-7_8.
[17] F. Matus and B. Hedblom, "Addressing the low-altitude airspace integration challenge—USS or UTM core?" in *Proc. Integr. Commun. Navig. Surveillance Conf. (ICNS)*, 2018, pp. 2F1-1–2F1-11, doi: 10.1109/ICNSURV.2018.8384848.
[18] A. Steinberg, M. Cardei, and I. Cardei, "UAS path planning using a space-time graph," in *Proc. IEEE SysCon*, Aug. 2020, pp. 1–8.
[19] A. Steinberg, M. Cardei, and I. Cardei, "UAS batch path planning with a space–time graph," *IEEE Open J. Intell. Transp. Syst.*, vol. 2, pp. 60–72, 2021, doi: 10.1109/OJITS.2021.3070415.
[20] "Python." 2021. [Online]. Available: https://docs.python.org(Accessed: Nov. 2021).
[21] "High performance computing, FAU HPC." [Online]. Available: https://hpc.fau.edu/(Accessed: Nov. 2021).
[22] "Ko'Ko, FAU HPC." [Online]. Available: https://hpc.fau.edu/resources-2/koko/ (Accessed: Nov. 2021).
[23] "The SLURM workload manager for Linux clusters." [Online]. Available: https://slurm.schedmd.com/ (Accessed: Nov. 2021).
[24] "Query features—OpenStreetMap." 2021. [Online]. Available: https://www.openstreetmap.org/query?lat=26.3678&lon=-80.0780#map=14/26.3497/-80.0777 (Accessed: Nov. 2021).
[25] D. Sacharny and T. C. Henderson, "A lane-based approach for large-scale strategic conflict management for UAS service suppliers," in *Proc. Int. Conf. Unmanned Aircraft Syst.*, Jun. 2019, pp. 1–9.