Abstract—We propose a novel framework for the allocation of drone swarms for delivery services known as Swarm-based Drone-as-a-Service (SDaaS). The allocation framework ensures minimum cost (aka maximum profit) to drone swarm providers while meeting the time requirement of service consumers. The constraints in the delivery environment (e.g., limited recharging pads) are taken into consideration. We propose three algorithms to select the best allocation of drone swarms given a set of requests from multiple consumers. We conduct a set of experiments to evaluate and compare the efficiency of these algorithms considering the provider’s profit, feasibility, requests fulfilment, and drones utilization level.

Index Terms—Drones swarm, Service composition, Swarm allocation, Homogeneous swarms, Provider-centric, Congestion-aware.

I. INTRODUCTION

Drone-as-a-Service (DaaS) is a concept used to describe the use of the service paradigm to model services offered by unmanned aerial vehicles [1]. These services include, but are not limited to, agriculture [2], geographic mapping [3], and package delivery [4]. Of particular interest, is the application of DaaS in delivery. Drone delivery has recently seen a boost in interest from the industry [5] and research [6] as the need for contact-less deliveries increases in times of pandemics. Governments are relaxing delivery restrictions and regulations which accelerates the drone delivery growth [5]. Swarm-based DaaS (SDaaS) augments the concept to describe drone services that are delivered by a swarm of drones instead of single drones [7]. SDaaS services are widely used in search and rescue [8], sky shows entertainment [9], airborne communication networks [10], and delivery [11].

Swarms of drones in delivery are able to overcome a single drone delivery limitations. A swarm may be used for the timely delivery of heavier and/or multiple packages which go beyond the capability of one single drone [7]. A swarm may also travel longer distances as the payload may be distributed amongst the drones which in return reduces the pressure on the battery [11]. In addition, recent USA’s Federal Aviation Administration (FAA) drone flying regulations approved the use of small drones for delivery (payload < 2.5 kg) [1]. Therefore, larger drones are not regulated as they are neither practical nor safe in a city which raises the need for drone swarms in delivery. Swarm-based Drone-as-a-Service for delivery is defined as the use of swarm of drones to carry multiple packages in a line of sight skyway segment [7]. The skyway segments connect nodes that maybe the source, destination, or intermediate nodes representing buildings rooftops equipped with recharging stations. The nodes and segments would make an SDaaS skyway network. SDaaS delivery composes of three main components. First, delivery swarms must be allocated for delivery requests. Second, the best path must be composed from the source to the destination that optimizes the delivery. Last, in case of failure, recovery plans must be implemented. In this paper, we focus on the first component (allocation) taking into consideration the effect on the second component (composition). Previous works focused on the composition assuming the existence of pre-allocated swarms [7].

Swarm-based drone delivery services lend themselves quite naturally to being modelled using the service paradigm because they map to the key ingredients of the service concept, i.e., functional and non-functional attributes [7]. The function of an SDaaS is the delivery of packages from a source to a destination. The non-functional aspects or the Quality of Services (QoS) include the delivery time, energy consumed, cost, etc. We address the allocation of SDaaS services from a provider perspective, i.e. optimizing the QoS for the benefit of the provider. We focus on optimizing the profit through the allocation of swarm members to consumer’s requests. The allocation of the swarm members is a core part of swarm-based services as it depends on the consumers requests and directly affects the optimal composition of SDaaS services.

There are several challenges that require addressing when allocating swarms for delivery requests. We assume that an SDaaS service provider owns a fleet of $n$ drones that are used to make multiple swarms to serve multiple requests. The swarms need to be allocated optimally to deliver requests bounded by strict time windows. A late or early delivery is considered unsuccessful. In addition, swarms may travel longer than a request’s time window for certain delivery missions (e.g. long distances). This causes overlapping travel times that may affect delivery requests in the next time window. Furthermore, congestion at recharging nodes may occur if multiple swarms compose the same path simultaneously. Given the aforementioned challenges, we propose to allocate SDaaS swarms to serve consumers...
requests optimally from a provider perspective. This work is the first to introduce the allocation and re-allocation problem of homogeneous drone delivery swarms to a set of time constrained requests using a service-oriented approach.

We summarize our main contributions as follow:

- Define a new set of SDaaS services allocation constraints.
- Propose a modified A* congestion-aware algorithm to compose SDaaS services.
- Propose three SDaaS allocation algorithms optimizing profit for the providers.

A. Motivating Scenario

Let’s assume that, within a day, several medical facilities request multiple packages from a supplier to be delivered together. Hence, multiple swarms are needed to deliver these packages to their respected destinations. Let’s also assume that each facility selects a time window for packages arrival. As shown in Fig. 1, requested packages may be of different weights. We assume that the maximum package weight does not exceed the drones payload limitations. The weight of the packages directly affects the energy consumption of the drones. This in return will affect the distance the drone packages can travel before the need to recharge. On the other hand, the medical supplier owns a finite set of drones that are used to make different swarms to serve the requests. The drones are homogeneous, i.e. same type. Therefore, the drones have same payload capacities, battery capacities, and power consumption rates. These drones would traverse through a line of sight skyway network (following drone fly regulations) till they reach the destinations. The intermediate nodes are buildings’ rooftops supplied with multiple recharging pads. Each node contains different numbers of recharging pads. We assume the environment is deterministic, i.e. we know before hand about the packages weights, availability of recharging pads, battery capacities, etc. As shown in the figure, if the swarm serving destination 26 spends more than an hour to deliver and return back to the source, the delivery to destination 35 will be affected as it needs to re-use a drone from the first swarm to deliver 4 packages.

Given r requests per day, how to allocate the homogeneous swarms from a finite fleet n to serve the different requests? What is the maximum number of requests could a fleet of size n undertake within a day? How to deal with overlapping travel times? How to allocate swarms to requests that are most profitable for providers? How to re-use and schedule the swarms to maximize the fleet utilization? Furthermore, how to compose the different swarms paths to reduce congestion in the network. Given all the aforementioned constraints, and without loss of generality, the system should ensure the arrival of the packages to the destinations within consumers’ specified time windows. Future work would extend this work to address environmental issues and uncertainties in the delivery environments.

II. RELATED WORK

Swarming can be defined as the collective behavior of a group of entities to achieve a common goal [12]. Examples in nature include bird flocks, fish shoals, and honey bees [13]. Similarly, drone swarms can be defined as a set of drones acting as a single entity to achieve a common goal [14]. Drone swarms are able to overcome single drones limitations and provide faster and more efficient services [15]. In this regard, drone swarms have enormous potential across a broad spectrum of civilian applications. The recent literature spotlights several applications that are already being tested and others envisioned for the future. This includes search and rescue [8], weather monitoring [16], and delivery [12].

In delivery, the majority of the literature refer to drone swarms as a set of individually operated drones managed to serve multiple independent deliveries [17] [18]. However, we define a swarm as a set acting as a single entity and travelling together to deliver multiple packages to a single destination. In this respect, a sequential and parallel drone swarm delivery services compositions was proposed [7]. A sequential composition is when a swarm travels together bounded by a space window from a source to a destination. A parallel composition occurs with dynamic swarms capable of splitting and merging between the source and the destination. In addition, the effect of swarm formation and shape on delivery services composition was explored [11]. However, to the best of our knowledge, no literature discussed the optimal allocation of drone swarms to serve multiple requests in a delivery context.

Multi-robot task allocation (MRTA) is an area concerned about the selection of the best robot, out of a set of robots, and allocating it to the best suitable task, out of a set of tasks, to optimize an overall system performance [19]. This area also deals with homogeneous and heterogeneous robots. Homogeneous robots are robots within a group with same capabilities [20]. Heterogeneous robots are robots within a group with different capabilities [21]. For example, homogeneous drones are drones with same speeds, sizes, or payload and battery capacities. The tasks that require multiple robots to be executed are referred to as Multi-robot Tasks [12]. In this regard, to the best of our knowledge, the allocation of homogeneous drone swarms to a set of constrained tasks has not yet been explored. In addition, the re-allocation and scheduling of swarms to perform multiple tasks has not been addressed. Thus, this work is the first to introduce the allocation and re-allocation
problem of homogeneous swarms (drone delivery swarms) to a set of time constrained tasks (consumers requests) using a service-oriented approach.

The service paradigm leverages the drone technology to effectively provision drone-based services \([1]\). The service paradigm presents a higher level of abstraction that provides congruous solutions to real-world problems \([22]\). Single drone delivery services framework was presented as Drone-as-a-Service (DaaS) \([1]\). This framework was later expanded to model swarm-based services called Swarm-based Drone-as-a-Service (SDaaS) as they map to the key ingredients of the service paradigm, i.e., functional and non-functional attributes \([7]\). Hence, this work extends the SDaaS framework to cover the key element of allocating the service swarm members to optimize the service’s QoS from a provider perspective. This work also considers SDaaS services composition and delivery delays that may be caused due to congestion and the concurrent use of the skyway network medium.

III. SWARM-BASED DRONE-AS-A-SERVICE MODEL

In this section, a service model for swarm-based delivery services is presented. We abstract each swarm travelling on a skyway between two nodes as an SDaaS (see Fig. \([1]\)). We formally define a Swarm-based Drone-as-a-Service (SDaaS). We then define an SDaaS provider and a consumer’s request. Later, we discuss the constraints surrounding the allocation of SDaaS services.

Definition 1: Swarm-based Drone-as-a-Service (SDaaS).

An SDaaS is defined as a set of drones, i.e. more than one, carrying packages and travelling in a skyway segment. It’s represented as a tuple of \(< SDaaS_{id}, S, F >\), where

- \(SDaaS_{id}\) is a unique service identifier
- \(S\) is the set of drones travelling in SDaaS. \(S\) consists of \(D\) which is the set of drones forming \(S\); a tuple of \(D\) is represented as \(\langle d_1, d_2, ..., d_m \rangle\). \(S\) also contains the properties including the current battery levels of every \(d\) in \(\langle b_1, b_2, ..., b_m \rangle\), the payloads every \(d\) in \(D\) is carrying \(\langle p_1, p_2, ..., p_m \rangle\), and the current node \(N\) the swarm \(S\) is at.
- \(F\) describes the delivery function of a swarm on a skyway segment between two nodes, A and B. \(F\) consists of the segment distance \(dist\), travel time \(tt\), charging time \(ct\), and waiting time \(wt\) when recharging pads are not enough to serve \(D\) simultaneously in node B.

Definition 2: SDaaS Service Provider. A provider is presented as a tuple of \(< D, \alpha >\), where

- \(D\) is the finite set of \(n\) drones owned by the provider. \(D\) is a tuple of \(\langle d_1, d_2, ..., d_n \rangle\) and every drone \(d_i\) consists of a tuple of \(\langle b, p, s \rangle\) where \(b\) is the maximum battery capacity of the drone, \(p\) is the maximum payload capacity of the drone, and \(s\) is the maximum speed of the drone.
- \(\alpha\) is the providers locations, i.e. source node.

Definition 3: SDaaS Request. A request is a tuple of \(< R_{id}, \beta, P, T >\), where

- \(R_{id}\) is the request unique identifier.
- \(\beta\) is the request destination node.
- \(P\) are the weights of the packages requested, where \(P\) is \(\langle p_1, p_2, ..., p_m \rangle\).
- \(T\) is the time window of the expected delivery, it is represented as a tuple of \(\langle st, et \rangle\) where \(st\) is the start time of the requested delivery window and \(et\) is the end time of the requested delivery window.

A. SDaaS members allocation constraints

We address the following constraints surrounding the swarm-based services allocation to deliver on its potential:

- **Strict time of delivery**: A delivery request is strictly bounded by a time window of packages arrival. This means if a package arrives before the window start time or after the window end time the delivery is considered unsuccessful. If a package arrives earlier than scheduled, a consumer might not be at the destination location to receive it. In a similar manner, if a package arrives late, a consumer might not be satisfied by the service.

- **Time overlapped requests**: There might be instances where multiple consumers requests are to be delivered within the same time window. Since the provider has a finite number of \(n\) drones, some overlapped requests may not be served. This is true if the number of packages exceed the number of available drones. In this case, the provider needs to select the most profitable requests and reject the least profitable ones. Fig.\([2]\) depicts this constraint were R1-R3 are overlapping and only two requests could be served at a time since the provider owns 5 drones only.

- **Inter-dependency of delivery requests**: The finite drones need to be allocated to the requests that will maximize the profit for the provider. Hence, the drones may be allocated multiple times to serve multiple requests. However, there might be instances when selecting a most profitable request within a time window may lead to a non-optimal profit (R1 in Fig.\([2]\)). This is because the long Round Trip Time (RTT) for the R1 swarm will lead to opting out R4 that is more optimal. In addition, selecting R4 allows the selection of R3 which would not have been selected if R1 is selected. Hence, the allocation should consider the time
inter-dependent requests to maximize the provider profit throughout the full day.

- **Skyway network contentions:** Congestion at nodes may occur since concurrent requests mean that multiple swarms will be using the skyway network at the same time. This will cause increased delivery times. Hence, we need to address the SDaaS service composition and path selection to reduce the number of sequential charges due to congestion.

- **SDaaS environment constraints:** In addition to all the aforementioned constraints, the authors of [7] have defined some constraints in the SDaaS environment. This includes the different battery consumption rates within a swarm due to different carried payloads. In addition, the number of drones in a swarm may exceed the number of available recharging pads at a node causing sequential charging.

Our aim is to successfully allocate and re-allocate swarms to serve the best set of consumers requests, from a provider’s perspective, given the aforementioned constraints. To the best of our knowledge, this paper is the first to propose the optimal allocation of swarm services to consumers’ requests.

IV. SDaaS MEMBERS SELECTION AND ALLOCATION FRAMEWORK

The SDaaS members selection and allocation framework is a provider-centric, profit-driven, framework. The framework consists of two main modules as shown in Fig. 3. The first is the congestion-aware SDaaS services composition (path composition) for every request. The output of this module is the maximum Round Trip Time (RTT) for every request and the profit if the request is allocated and served. The second module uses the maximum RTT produced by module one and allocates the drones to the requests to maximize the utilization of the provider’s drones and increase the providers profit.

A. Congestion-aware SDaaS Composition

This paper focuses on a provider owning a finite set of homogeneous drones, i.e. drones with same capabilities. We assume that the packages in a request do not exceed the payload capacities of the drones. In addition, a drone can carry one package at a time and serve one request in a single trip. Therefore, the number of drones in a swarm, serving a request, is equal to the number of packages in a request. We also assume a request may consist of maximum \( m \) packages, i.e. maximum \( m \) drones form a swarm. The goal of this module is to compute the maximum time taken by a trip taken by a swarm to serve a request and return back to the origin RTT. Computing the RTT for a request is important in order to reuse the drones in other requests. The RTT is highly affected by the path taken by the swarm to the destination and its return path. The number of recharging pads, as described earlier, is different at each node. Some nodes may have recharging pads that are less than the number of drones in the swarm. Hence, sequential charging may occur adding to the RTT. In addition, contention may occur at a node if two swarms serving different requests taken the same path at a time causing congestion. Hence, we propose a modified A* congestion-aware SDaaS services composition algorithm. The RTT along with the number of drones used determine the profit of the provider if the request is served. We assume that the environment is deterministic, i.e. we know the availability of recharging pads considering other providers using the network. All the algorithms are presented below.

**Algorithm 1 Congestion-aware Services Composition Algorithm**

**Input:** \( S, R \)

**Output:** \( RTT, \text{Profit} \)

1: \( RTT = 0 \)

2: while \( S \) is not at destination and back at the source do

3: \( \text{distance to destination} = \text{Dijkstra}(\text{current, destination}) \)

4: \( \text{compute energy consumption for every } d \text{ based on } R \text{ package weights and distance to destination} \)

5: if all \( d \) in \( S \) can reach destination without intermediate nodes then

6: \( S \text{ travels to destination} \)

7: \( t+\text{travel time} \)

8: else

9: \( \text{find all nearest neighbor nodes} \)

10: \( \text{if } P_D - S_D < m \text{ then} \)

11: \( \text{available_pads=total_pads - (} P_D - S_D \text{)} \)

12: \( \text{else} \)

13: \( \text{available_pads=total_pads - } m \)

14: end if

15: \( \text{select best neighboring node (min travel time and min node time)} \)

16: \( S \text{ travels to neighboring node} \)

17: \( RTT+=\text{travel time + charging time + waiting time} \)

18: end if

19: end while

20: \( \text{Profit} = \text{size}(S_D) \times RTT \times \text{Constant} \)

21: return \( RTT, \text{Profit} \)

Algorithm 1 illustrates the steps for finding the best neighboring node to reach the destination. The profit calculated is based on the size of the swarm and the travel time. The algorithm iterates through all the nearest neighbor nodes to find the best one that satisfies the constraints and maximizes the profit. The profit is calculated by multiplying the size of the swarm by the RTT and a constant factor. The algorithm returns the RTT and the calculated profit.
the number of available recharging pads under congestion. We assume no more than two swarms may select a node at a time. The $RTT$ is updated and the drone again checks its potential to reach the destination. The process repeats until the swarm is at the destination. Once at the destination, the swarm charges full and travels back to the source in the same manner. The energy consumption rate at the return trip is much lower than the outbound trip due to the released payload. The $RTT$ is updated at the source after charging fully. Once the $RTT$ is computed the profit of the request, if allocated and served, is computed using the $RTT$ and the number of drones serving the request $S_D$. The profit is computed per mile per drone taking the constraints into consideration. Algorithm [1] describes the congestion-aware services composition process.

B. SDaaS members Allocation and Requests Scheduling

The output of the first module, i.e. the maximum $RTT$ and Profit, for all requests is used to allocate the drones swarm members to the most profitable combination of requests. In this section, we propose three methods for the swarm-request allocation. We assume that the requests are received in batch for the full day. We divide the day into $t$ time windows $W$. For simplicity, we assume that a drone may be used once in a time window. Hence, if a request’s $RTT$ overlaps over two time windows, we assume that the swarm serving this request is booked for both time windows. This assumption would be relaxed in the future work. We need to allocate the swarms to serve the most profitable requests while maximizing the utilization of the $finite$ number of drones that the provider owns.

The swarms-requests allocation problem is a part of the Multi-Robot tasks allocation problems. However, to the best of our knowledge, no previous work has addressed the allocation problem from a service provider perspective to allocate a $finite$ set of robots to time-constrained consumers requests (tasks) and ensuring the re-use of robots to maximize their utilization. Each drone may serve more than one request at different times. We propose to tackle this problem through three proposed algorithms namely: Time-based greedy algorithm, Request-based greedy algorithm, and the Heuristic-based algorithm. Later, we will compare the performance of these algorithms against the Brute force algorithm.

1) Request-based greedy allocation algorithm: In this approach, all the requests are first sorted descendingly by their profit. The algorithm allocates swarms to the requests in order of profit if the request is servable with the available drones. Algorithm [2] describes the request-based greedy algorithm. $P_D$ is the $finite$ number of drones owned by the provider. The requests are looped through once. Each request checks for $two$ conditions. The first condition checks if there are available drones within its window ($W_{R_i}$) to serve the request [line 4]. This is done by checking if the number of the request’s drones ($D_{R_i}$) plus the number of used (occupied) drones ($used_drones[W_{R_i}]$), within the same time window, are less than or equal to the total owned drones of the provider ($P_D$). The second condition is checked if the first condition is true. Here, the algorithm checks if the request’s $RTT$ overlaps over the next time window ($W_{R_i + 1}$) [line 5]. In this case, if there are available drones in the next time window the swarm gets allocated to serve the request ($served_R$). The allocated request’s profit ($P_{R_i}$) is added to the $total$ profit, the request’s drones ($D_{R_i}$) are added to the used drones of the two time windows ($used_drones[W_{R_i}, W_{R_i + 1}]$) and to the total utilized drones $drones_used$. Otherwise, if there are no available drones in the next time window $W_{R_i + 1}$ the request is discarded. If the second condition is not satisfied [line 12], i.e. the request does not overlap over the next time window, the swarm is allocated to serve the request, and are added to the $used_drones[W_{R_i}]$ of the first time window only. The process continues until all requests are looped through. The sum of the profits of the served requests and the drones used in them is returned at the end.

2) Time-based greedy allocation algorithm: This approach is a slight modification to the request-based greedy algorithm where the request time window is taken in consideration when sorting the requests. Here, all the requests are first sorted by their start times ascendingly. All the requests within a time window are then sorted by their profit descendingly. The rest of the algorithm checks the validity of the requests combination similar to Algorithm [2] and computes the total profit, drones utilized, and the set of the served and allocated requests.

3) Heuristic-based allocation algorithm: The main limitation of the greedy algorithms is the order of the requests
Algorithm 2 Request-based greedy allocation algorithm

```
Input: R, P_D
Output: served_R, total_profit, drones_used
1: total_profit = 0, drones_used = 0
2: served_R = [], used_drones = []
3: sorted_R = sort_by (R, profit)
4: for each R in sorted_R do
5:   if (D_R + used_drones(W_{R_i})) <= P_D then
6:     if (D_R + used_drones(W_{R_i} + 1)) == True then
7:       served_R.append(R_i)
8:       total_profit += P_{R_i}
9:       used_drones[W_{R_i}, W_{R_i} + 1] += D_{R_i}
10:      drones_used += D_{R_i}
11:     end if
12:   else
13:     total_profit += P_{R_i}
14:     used_drones[W_{R_i}], += D_{R_i}
15:     served_R.append(R_i)
16:     drones_used += D_{R_i}
17:   end if
18: end for
19: return (served_R, total_profit, drones_used)
```

Algorithm 3 Heuristic-based allocation algorithm

```
Input: R, P_D
Output: served_R, total_profit, drones_used
1: all_profit = [], all_drones_used = [], all_served_R = []
2: for each r in R do
3:   active_total_profit = 0, active_served_R = [],
4:   active_used_drones = [], active_drones_used = 0
5:   for each AR in loop around (r.id) do
6:     if (D_{AR_i} + used_drones(W_{AR_i})) <= P_D then
7:       if overlap (RTT_{AR_i}, W_{AR_i} + 1) == True then
8:         active_served_R.append(A_{R_i})
9:         active_total_profit += P_{AR_i}
10:        active_used_drones[W_{AR_i}, W_{AR_i} + 1] += D_{AR_i}
11:       active_drones_used += D_{AR_i}
12:     end if
13:   else
14:     active_total_profit += P_{AR_i}
15:     active_used_drones[W_{AR_i}], += D_{AR_i}
16:     active_served_R.append(A_{R_i})
17:     active_drones_used += D_{AR_i}
18:     end if
19: end if
20: end for
21: all_profits.append(active_total_profit)
22: all_drones_used.append(active_drones_used)
23: all_served_R.append(active_served_R)
24: end for
25: total_profit = max(all_profits)
26: max_index = all_profits.index(max_profit)
27: served_R = all_served_R[max_index]
28: drones_used = all_drones_used[max_index]
29: return (served_R, total_profit, drones_used)
```

processed and served. Sorting the requests on profit and time does not ensure the most optimal solution. This approach is, hence, a modification of the greedy algorithms that tries to tackle the order limitation. Algorithm 2 depicts the heuristic-based allocation method. In this approach, the unsorted list of requests are looped through considering different allocation start points resulting in different combinations of requests. Two main set of arrays and variables are constructed. The first set of arrays are of the size of the requests R storing the profits (all_profits), drones utilized (all_drones_used), and the combination of served requests (all_served_R) generated by different starting points (starting request). The second set of arrays and variables are local temporary active arrays that are reconstructed at every starting point (request). These second set of arrays and variables store the total profit (active_total_profit), drones utilized (active_drones_used), and the served requests combination (active_served_R) with a certain starting point (request). A nested loop is constructed where the outer loop loops over all requests R [line 2]. The inner loop, similar to the greedy, loops over the requests and checks the valid combination however starting from the request which the outer loop is at (r.id) [line 4]. After every iteration of the inner loop, the variables of the valid served requests are updated in the active set of arrays and variables. After the full iteration of the inner loop, the first set of arrays, of size R, store the total profit, drones utilized, and the combination of served requests at the index of the starting point (request) (r.id). Once the outer loop finishes, the algorithm finds the combination of requests with the maximum profit and the number of drones utilized in this combination [lines 25-28].

Fig. 4 shows an allocation example of the three proposed algorithms with a set of 10 requests (0-9). The bars represent the maximum RTT for each request. In this example, we assume that the provider owns 6 drones only. The drones are used to make up swarms to serve the different requests. As mentioned earlier, we assume that a drone once used in a time window may not be used again within the same window. The algorithms result in a set of selected requests that the drones get allocated to. As shown in the figure, the heuristic-based algorithm outperforms the two methods with a total profit of 272.65 and 7 served requests out of 10. Next, the request-based greedy algorithm served 6 requests with a total profit of 252.9. Finally, the time-based greedy algorithm served 5 requests with a total profit of 211.91.
The main down-side of the time-based greedy algorithm is the starting time window. If requests are served and allocated in time window \( i \) use all/most drones and the served requests overlap to the next time window \((i+1)\), there are chances that more profitable requests in the next time window will not be served. As shown in Fig. 4, since requests 1 and 3 have been allocated, as they arrive earlier, the more profitable requests 2 and 9 were not served. In the same manner, the request-based greedy algorithm chooses the most profitable request but there might be a combination of other smaller requests with a higher profit. As shown in the figure, the algorithm selects request 8 to be served whereas the total profit of requests 5 and 6 is higher. Since the heuristic-based algorithm overcomes the limitation of requests order, it smartly outperforms the two other methods and allocates swarms to serve requests that might be less profitable but in total resulting in more profit and more served requests.

## V. Experiments and Results

In this section, we evaluate the performance of the three proposed methods of allocations. We conduct a set of experiments to evaluate the performance in terms of profit maximization, drones utilization, requests fulfillment, and computation. Although there are existing resource allocation algorithms [25] [26], the proposed problem is fundamentally different in that it is machine (drone) unspecific, re-allocatable, and time dependant. Hence, we compare the proposed methods against the allocation-optimal and exhaustive Brute force allocation to measure their performance.

The dataset used in the experiments is an urban road network dataset from the city of London. The data consist of a graph edge list of the city [27]. For the experiments we took a sub-network of 129 connected nodes to mimic a skyway network. Each node was allocated with different number of recharging pads. We then set a source node and generate \( r \) requests with different destination nodes. For each request, we synthesize payloads with a maximum size of 5 packages and a maximum weight of 1.4 kg. We use the congestion-aware SDaaS composition algorithm to compute the profit and the maximum RTT of each request given the different number of recharging pads at each node. Each generated request is randomly assigned to a delivery time window. We assume the drone takes 30 minutes to fully charge and travel time is computed using the distance of each segment in the composed path. The rest of the experimental variables are presented in Table I.

In the first experiment, we compare the performance in terms of profit of the three proposed algorithms against the optimal and exhaustive Brute force allocation approach. In the brute force approach, all possible combination of requests are generated. Combinations that are non feasible due to the finite number of provider’s drones are discarded. The feasible combination with the highest profit is returned and the swarms are allocated to the requests of this combination. In this experiment, we assume a provider owns 30 drones that are allocated and re-used. As shown in Fig. 5, the heuristic-based approach outperforms the request-based and time-based greedy algorithms. The x-axis represents the number of requests received within a day. The y-axis represents the total profit made by serving the allocated requests. The brute force approach, as shown in the figure, terminated at 27 requests. This is because the memory usage is highly exponential as all possible combination of requests are generated and processed. This shows that the brute force approach is not feasible in real-word scenarios where the number of delivery requests received a day may be large in number. The heuristic-based approach outperforms the request-based and time-based greedy algorithms. As explained earlier, this is due to the heuristic-based algorithm capability of overcoming the order of allocation constraint (it does not allocate the most profitable requests first as shown in Fig. 4. The heuristic-based algorithm smartly outperforms the two other proposed methods and allocates swarms to serve requests with less profit but in total resulting in more profit and more served requests. The request-based and time-based greedy algorithms are interchangeably preforming better than each other. This, as described earlier, mainly depends on the order of the received requests. In some cases, where requests with higher profits are widely distributed over the time windows, the time-based method would perform better. Otherwise, if most profitable requests are not distributed, the request-based method performs better.

Although the heuristic-based method is not always optimal, it is the best performing feasible method. The heuristic-based

| Variable                                      | Value                                      |
|-----------------------------------------------|--------------------------------------------|
| PC information                               | 7th Gen Intel Core™ i7-7700HQ Processor (2.8 GHz) |
| No. of nodes in the largest connected sub-network | 195                                        |
| Max no. of packages in a request              | 5                                          |
| Max weight of a package                       | 1.4 kg                                     |
| Drone model                                   | DJI Phantom 3                              |
| Time for drone to fully charge                | 30 mins                                    |
| Battery capacity                              | 4480 mAh                                   |
| Drone speed                                   | 15.6 m/s                                   |

![Fig. 5. Profit with varying number of received requests](image-url)
Fig. 6. Requests allocation of Brute Force Vs Heuristic (provider owned drones=6)

Fig. 7. Execution times of proposed methods

Fig. 8. Successful requests varying the number of received requests

Fig. 9. Successful requests varying the number of owned drones

A method in few cases does not give the optimal solution when compared to the brute force approach. Fig. 6 shows an example of such a situation. As shown in the figure, the only difference between the heuristic and brute force is the allocation of requests 2 (more profitable) and 3. Since the heuristic loops over the requests in sequence, changing the starting point at every iteration, there are chances that requests with lower profits get allocated and requests with higher profits coming later do not find drones to be served. The selected requests in the figure started at iteration 3, i.e. request 3 was the first to get allocated. Since requests 3, 7, and 9 got allocated (in sequence) with the heuristic method, requests 1 and 2 did not have enough drones in their respected time windows to get allocated. Regardless of the limitation described, the execution time of the heuristic-based algorithm is polynomial compared to the exponential time of the brute force making it a more scalable and feasible solution. Fig. 7 shows the execution times of all methods varying the number of requests received a day. The left y-axis represents the execution time for the brute force algorithm. The right y-axis represents the execution times of the three proposed methods.

In the second experiment, we study the performance of the proposed algorithms on the percentage of fulfilled requests. A fulfilled request is a request that is successfully allocated to a swarm of drones to be served. Fig. 8 shows the percentage of successfully fulfilled requests as the number of requests increase in a day. We assume that the provider owns a set of 30 drones only. As shown in the diagram, when the number of requests are low the provider’s drones are able to serve all the requests. As the requests received per day increase, the percentage of successful requests decreases. This is because the finite drones limit the number of requests that could be served. The heuristic-based approach distinctly outperforms the greedy approaches and is capable of serving more requests a day. This is reflected by the profit the provider gains (Fig. 5) and will lead to more customer’s satisfaction as more requests are served. Such graph, allows the provider to determine the number of requests they should receive a day if they own a finite number of drones and are incapable of enlarging the fleet. Fig. 9 measures the percentages of fulfilled requests with a fixed number of requests (50) and varied number of provider’s owned drones. The graph shows how the heuristic-based method outperforms the other two methods. As the number of owned drones increase more requests are getting fulfilled. Such graph, helps the provider to determine the optimal number of drones they should own if they receive a daily $r$ number of requests.

In the last experiment, we measure the effectiveness of the methods in terms of drones utilization. A provider, as described earlier, owns a finite set of drones. An optimal
method is a method that re-uses these drones as much as possible to serve multiple requests. Fig. 10 shows the percentage of utilized drones, i.e. the number of times the drones are re-allocated, as the number of requests received increase. We assume that the provider owns 30 drones. As shown in the figure, as the number of received requests increase, the percentage of drone’s utilization increases. The percentage stabilizes when all the drones are fully occupied at all time slots. As shown in the graph, the heuristic method outperforms the other methods in terms of drones utilization. The drones at requests 110-200 are used more than five times in a day with seven time windows and window overlapping requests. Fig. 11 shows the percentages of drones utilized as the number of owned drones increases with a fixed number of received requests a day (50). As the number of owned drones increases the drones utilization decreases as they will be re-allocated less. The heuristic-based method, as shown, outperforms the other two methods in terms of drones utilization. At number of drones 5 the heuristic-based method and the request-based greedy algorithms utilize the same number of drones. However, if we look at Fig. 9 the percentage of fulfilled requests with 5 drones is less with the request-based greedy method. This means the request-based greedy method is using more drones to serve less number of requests. Using the drones utilization graphs along with the graphs of successful requests, the provider may determine the optimal number of drones they should own to serve maximum number of requests while utilizing the drones as much as they can without increasing the cost on themselves.

VI. CONCLUSION

We proposed a provider-centric allocation of drone swarm services known as, Swarm-based Drone-as-a-Service (SDaaS). A congestion-aware SDaaS composition algorithm is proposed to compute the maximum round trip times a swarm may take to serve a request taking the constraints at intermediate nodes (limited recharging pads and congestion) in consideration. Three swarms-requests allocation methods were proposed with the goal of increasing the provider’s profit namely; request-based greedy, time-based greedy, and heuristic-based allocations. The efficiency of the proposed approaches was evaluated in terms of profit maximization, execution times, requests fulfilment, and drones utilization. The efficiency of the proposed approaches where compared to a brute force baseline approach and has shown their feasibility on scaling compared to the brute force. Experimental results show the outperformance of the heuristic-based approach in comparison with the other two algorithms and its near optimal results. The limitation of the heuristic-based method is also described and compared to the brute force method. In the future work, we will consider heterogeneous swarms allocation to serve multiple requests and extend the work to deal with SDaaS failures.

REFERENCES

[1] B. Shahzaad, A. Bouguettaya, S. Mistry, and A. G. Neiat, “Resilient composition of drone services for delivery,” FGCS, 2021.
[2] U. R. Mogili and B. Deepak, “Review on application of drone systems in precision agriculture,” Procedia computer science, vol. 133, 2018.
[3] S. Madawalagama, N. Munasinghe, S. Dampegama, and L. Samarakoon, “Low cost aerial mapping with consumer grade drones,” in 37th Asian Conference on Remote Sensing, 2016.
[4] B. Shahzaad, A. Bouguettaya, and S. Mistry, “A game-theoretic drone-as-a-service composition for delivery,” in 2020 ICWS, IEEE.
[5] W. E. Forum, “We’re about to see the golden age of drone delivery – here’s why,” Jun 2020.
[6] J. Euchi, “Do drones have a realistic place in a pandemic fight for delivering medical supplies in healthcare systems problems,” Chin. J. Aeronaut., 2020.
[7] B. Alkouz, A. Bouguettaya, and S. Mistry, “Swarm-based drone-as-a-service (sdaas) for delivery,” in 2020 IEEE International Conference on Web Services (ICWS), pp. 441–448, IEEE, 2020.
[8] G. A. Cardona and J. M. Calderon, “Robot swarm navigation and victim detection using rendezvous consensus in search and rescue operations,” Applied Sciences, vol. 9, no. 8, p. 1702, 2019.
[9] M. Waibel, B. Keays, and F. Augugliaro, “Drone shows: Creative potential and best practices,” tech. rep., ETH Zurich, 2017.
[10] X. Cao, P. Yang, M. Alzenad, X. Xi, D. Wu, and H. Yanikomeroglu, “Airborne communication networks: A survey,” IEEE Journal on Selected Areas in Communications, vol. 36, no. 9, pp. 1907–1926, 2018.
[11] B. Alkouz and A. Bouguettaya, “Formation-based selection of drone swarm services,” EAI Mobiquitous Conference, 2020.
[12] S. Thornton and G. E. Gallasch, “Swarming logistics for tactical last-mile delivery,” in International Conference on Science and Innovation for Land Power, vol. 2018, 2018.
[13] J. van Ast, R. Babulka, and B. De Schutter, “Particle swarms in optimization and control,” IFAC Proceedings Volumes, 2008.
[14] W. Chen, J. Liu, H. Guo, and N. Kato, “Toward robust and intelligent drone swarm: Challenges and future directions,” IEEE Network, 2020.
[15] O. Shrit, S. Martin, K. Alagha, and G. Pujolle, “A new approach to realize drone swarm using ad-hoc network,” in *16th Annual Mediterranean Ad Hoc Networking Workshop*, pp. 1–5, IEEE, 2017.

[16] H. Hildmann, E. Kovacs, F. Saffre, and A. Isakovic, “Nature-inspired drone swarming for real-time aerial data-collection under dynamic operational constraints,” *Drones*, vol. 3, no. 3, p. 71, 2019.

[17] K. T. San, E. Y. Lee, and Y. S. Chang, “The delivery assignment solution for swarms of uavs dealing with multi-dimensional chromosome representation of genetic algorithm,” in *IEEE 7th Annual UEMCON*, pp. 1–7, IEEE, 2016.

[18] K. Kuru, D. Ansell, W. Khan, and H. Yetgin, “Analysis and optimization of unmanned aerial vehicle swarms in logistics: An intelligent delivery platform,” *IEEE Access*, vol. 7, pp. 15804–15831, 2019.

[19] A. Khamis, A. Hussein, and A. Elmogy, “Multi-robot task allocation: A review of the state-of-the-art,” *Cooperative Robots and Sensor Networks* 2015, pp. 31–51, 2015.

[20] O. Gigliotta, “Equal but different: Task allocation in homogeneous communicating robots,” *Neurocomputing*, vol. 272, pp. 3–9, 2018.

[21] M. Dorigo, D. Floreano, L. M. Gambardella, F. Mondada, S. Nolfi, T. Baaboura, M. Birattari, M. Bonani, M. Brambilla, A. Brutschy, *et al.*, “Swarmanoid: a novel concept for the study of heterogeneous robotic swarms,” *IEEE Robotics & Automation Magazine*, 2013.

[22] A. Bouguettaya, M. Singh, M. Huhns, Q. Z. Sheng, H. Dong, Q. Yu, A. G. Neiat, S. Mistry, B. Benatallah, B. Medjahed, *et al.*, “A service computing manifesto: the next 10 years,” *Communications of the ACM*, vol. 60, no. 4, pp. 64–72, 2017.

[23] R. N. Akram, K. Markantonakis, K. Mayes, O. Habachi, D. Sauveron, A. Steyven, and S. Chaumette, “Security, privacy and safety evaluation of dynamic and static fleets of drones,” in *IEEE/AIAA 36th DASC*, 2017.

[24] E. W. Dijkstra *et al.*, “A note on two problems in connexion with graphs,” *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[25] J. Carlier and É. Pinson, “An algorithm for solving the job-shop problem,” *Management science*, vol. 35, no. 2, pp. 164–176, 1989.

[26] H. M. Salkin and C. A. De Kluvyer, “The knapsack problem: a survey,” *Naval Research Logistics Quarterly*, vol. 22, no. 1, pp. 127–144, 1975.

[27] A. Karduni, A. Kermanshah, and S. Derrible, “A protocol to convert spatial polyline data to net. formats and apps to world urban road nets.,” *Scientific data*, vol. 3, p. 160046, 2016.