Joint Edge Resource Allocation and Path Planning for Drones with Energy Constraints

Giorgos Polychronis(✉) and Spyros Lalis

University of Thessaly, Volos, Greece
{gpolychronis,lalis}@uth.gr

Abstract. Several applications use drones as mobile sensors which can fly directly over the points of interest with minimal human intervention. In some cases, the data that is captured at a given point has to be processed before moving to the next one. Even though, in the spirit of edge computing, such computations can be offloaded to nearby servers, this becomes challenging when edge servers have limited resources and drones have limited operational autonomy. In this paper, we propose an algorithm that jointly plans the paths for the drones and allocates the available edge resources between drones in a fair way, while respecting such constraints. We evaluate our algorithm through a wide range of experiments and find that it can significantly reduce the mission times with no offloadings, by up to almost 28% while performing close to the ideal case where offloading is always possible.

Keywords: Drones · Resource allocation · Computation offloading · Path planning · Edge computing

1 Introduction

Aerial unmanned vehicles, also referred to as drones, have become a valuable tool for many different applications. In particular polycopter drones are very popular thanks to their agility and ability to fly/hover, takeoff and land vertically, making it possible to gather data via their onboard sensors in a flexible way.

In this work, we focus on applications where the drone has to visit specific points of interest, take measurements using its onboard sensors, and perform some computation on the data. We assume that each computation must be performed in situ, before moving to the next point of interest. For instance, it may be desirable to take some special action depending on the result of the computation. For heavyweight computations, the overall mission time can be reduced,

This research has been co-financed by the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH - CREATE - INNOVATE, project PV-Auto-Scout, code T1EDK-02435.

S. Longfei and P. Bodhi (Eds.): MobiQuitous 2022, LNICST 492, pp. 378–399, 2023.
https://doi.org/10.1007/978-3-031-34776-4_20

significantly, by exploiting edge servers located near the points of interest. However, this may not always be possible if several drones wish to use the same servers at the same time, in which case it is important to allocate the available server resources among the competing drones in a fair way.

Another challenge, which affects such resource allocation decisions, is the typically limited operational autonomy of small drones that run on batteries. In the general case, this is not sufficient to conduct large missions, especially ones that may involve time-consuming processing as discussed above. Therefore, drones need to perform one or more intermediate stops to switch batteries before they can proceed to execute the rest of the mission.

We tackle the problem in a holistic way, by jointly building paths that can be safely followed by the drones despite their energy constraints and fairly allocating the available edge resources so that the mission time of each drone is fairly reduced with respect to others. The main contributions of our work are: (i) we capture the above problem in a formal way; (ii) we propose an algorithm that builds paths under energy constraints while at the same time making a fair allocation of the edge resources among drones so that they evenly reduce their mission time; (iii) we evaluate the performance and robustness of the proposed algorithm for different battery switching delays and degrees of autonomy, showing that it can reduce the mission times with no offloadings, by up to almost 28% while performing close to the ideal case where offloading is always possible.

The rest of the paper is structured as follows. Section 2 discusses related work. Section 3 provides a formal description of the system model and problem. Section 4 presents the proposed algorithm, which is evaluated in Sect. 5. Finally Sect. 6 concludes the paper.

2 Related Work

Various works explore the efficient offloading of a task from nodes to more powerful computing infrastructures. In [7], the authors deal with the problem where a drone has to stop at predefined waypoints and perform heavyweight computations. To reduce the mission time, instead of performing the computation onboard, the drone opportunistically offloads its computations when it is in range of an edge server and the expected computation time is better than the local one. Another work that studies dynamic offloading decisions is [9], where a drone-based system is used for tracking moving objects. The authors try to minimize the response time by deciding whether to offload a computation or execute it on the drone locally. The authors of [5], propose a scheme for deciding the offloading of computations at the edge or the cloud. In [17], the main idea is to offload tasks that require intensive computations to the cloud, and communication-heavy tasks to the fog. In both cases, tasks are prioritized following an earliest deadline first (EDF) policy. The authors of [6] propose an algorithm for offloading decisions and resource allocation for multiple UAVs. Their goal is to minimize the average service latency.

As a key difference with the above, our work jointly addresses computation offloading and path planning. Also, we allocate the edge resources among drones to evenly reduce their mission time, instead of (just) minimizing service latency.

In many works, drones are used as edge nodes that serve mobile users. Such a problem is studied in [19] where a UAV is used as a base station for providing video content to multiple users. The objective is to plan the trajectory of the drone and the associated resource allocation to provide the required QoS and minimize the drone's operation time. In [18] the authors investigate a problem where mobile IoT devices offload their computation to a UAV. The goal is to make decisions about the offloading, resource allocation and UAV trajectory, so that the overall energy needed for the IoT devices to complete their tasks is minimized. Both works follow a similar algorithmic approach, which is to split the problem in smaller sub-problems that are solved iteratively until the value of the optimization criterion converges. In [20], a UAV is used by multiple mobile devices to offload their tasks, with the goal to minimize the average weighted energy consumption of the UAV and the mobile devices, by optimizing the trajectory of the UAV, the allocations of computation resources and the offloading decisions. [2] investigates a similar problem, for a UAV that is used to serve vehicles in a highway. The authors of [3] study the use of multiple drones for serving IoT devices. In this case, the problem is to optimize the number of drones, their association with specific locations so as to best serve the devices and the drone paths.

Unlike the above works that consider drones as mobile servers, in our case the drones are the clients that need to offload their tasks to stationary edge servers. From a high-level algorithmic perspective, we employ a similar approach to some of these efforts. Namely, our algorithm also separates the path optimization and the resource allocation dimension and deals with each problem separately, while optimizing the joint solution in an iterative way.

There is a large body of work targeting different path planning problems for drones, also considering their energy constraints. We briefly discuss some indicative efforts. In [4], multiple drones are controlled by different ground stations in order to perform a set of tasks under certain constraints, such as fuel tank capacity and maximum number of drones served per ground station, while optimizing various objectives, like the total fuel consumption, the number of drones used and the total travel distance and mission time. The authors of [13] consider drones carrying supplies to different locations after a disaster incident. Apart from the energy constraints, drones have payload limitations and must return to depot stations to reload. The goal is to minimize the total distance, time or cost of the routes while supplying the target locations giving priority to the ones with the greater needs. A dynamic pickup and delivery problem is studied in [10] for drones that can switch batteries at depot stations, with the objective to produce safe flight plans that minimize delivery time and unnecessary aerial movement.

Similar to the above works, we propose a path-planning heuristic for drones with energy constraints. However, the main difference is that we do this in tandem with resource allocation at the edge so that drones can exploit the available

servers in order to perform the required computations faster and perhaps even save some intermediate depot visits, leading to a reduced mission time.

Another well studied topic are mobile wireless sensor networks (WSNs). In these scenarios, the aspect of mobility concerns the sinks and/or sensors of the WSN [16]. The case of mobile sinks is more similar to our problem, especially regarding the planning of their movement inside or around the sensor network. A survey of such work is given in [8]. The main difference between mobile WSNs and our work is the key objective. Mobile WSNs typically focus in prolonging the lifetime of the system, improving the coverage or reducing the data delivery latency. In contrast, our work focuses on the exploitation of edge computing to accelerate processing and minimize the mission time in a fair way for all drones.

3 System Model and Problem Formulation

3.1 Drones, Paths and Flight Model

Let there be M drones, $d_m, 1 \leq m \leq M$. Each drone is assigned an independent mission, in which it must visit a set of waypoints $\mathcal{V}_m$. These waypoints represent points of interest where the drone has to perform some sensing and then process the collected data on the spot, e.g., to take further action if a problem is detected.

To visit the points of interest assigned to it, each drone d_m follows a path P_m, which is encoded as a sequence $P_m[i], 1 \leq i \leq \text{len}(P_m)$. The path must include all points of interest exactly once, $\exists! i : 1 \leq i \leq \text{len}(P_m) : P_m[i] = p, \forall p \in \mathcal{V}_m$. Also, each drone d_m initially starts its mission by taking off from a depot dep_m, and ends its mission by returning and landing at dep_m. We encode this in the drone's path by letting $P_m[1] = P_m[\text{len}(P_m)] = dep_m$. Without loss of generality, we assume the depot does not coincide with a point of interest, i.e., $dep_m \notin \mathcal{V}_m$. As will be discussed in the sequel, a drone may perform additional depot visits in order to switch batteries. Such intermediate depot visits, also referred to as depot detours, are explicitly encoded in the drone's path by inserting the depot as an intermediate waypoint.

Our flight model assumes polycopter drones, which have vertical take-off and landing capability, can fly between waypoints in a straight line, and can hover over a given position. Let $flyhT_m^{i,i+1}$ be the time that is needed for d_m to fly horizontally between $P_m[i]$ and $P_m[i+1]$. Further, let $takeoffT_m$ be the time needed for d_m to take-off from dep_m and reach the flight altitude for the mission at hand, and let $landT_m$ be the time needed to vertically land back at dep_m. These extra overheads are taken into account to derive the total flight delay between two successive waypoints in the drone's path, as follows:

$$flyT_m^{i,i+1} = \begin{cases} flyhT_m^{i,i+1}, & \text{if} P_m[i], P_m[i+1] \neq dep_m \\ flyhT_m^{i,i+1} + takeoffT_m, & \text{if} P_m[i] = dep_m \\ flyhT_m^{i,i+1} + landT_m, & \text{if} P_m[i+1] = dep_m \end{cases} \quad (1)$$

Note that the case where $P_m[i] = P_m[i+1] = dep_m$ is not handled above. This would mean that the drone takes-off from the depot only to immediately land back there, which cannot hold in a properly formed flight plan.

3.2 Sensing and Computation

At each point of interest, the drone takes some measurements via its onboard sensors, and then processes this data. We assume that each drone performs the same type of sensing and data processing at all points of interest. However, different drones may perform different types of sensing and data processing. Also, we assume that data processing needs to be performed in situ, while the drone is positioned over the point of interest, before moving to the next waypoint.

Let $senseT_m$ be the time that is needed for d_m to perform the required sensing. Also, let $comp_m$ be the type of computation that needs to be performed on this data, taking $data_m^{in}$ amount of data as input and returning $data_m^{out}$ amount of data as a result. Notably, the type of computation determines the amount of resources res_m^{req} needed to run the computation on a computing platform.

Onboard Computation. Each drone d_m has an onboard computer with hardware platform hw_m, which can be used to perform $comp_m$ locally. In order for the drone to be truly autonomous, its onboard platform has sufficient available resources to run the computation in question, $res_m^{avl} \geq res_m^{req}$. In this case, we assume that (local) data movement for $data_m^{in}$ and $data_m^{out}$ takes a negligible amount of time. Thus the total computation delay when processing is performed locally on the drone is $compT_m = procT(comp_m, hw_m)$.

Computation Offloading. As another option, the drone can offload its computation to a nearby server located at the edge in order to accelerate processing and reduce the mission time. We assume that each edge server is suitably prepared to provide such a computation as a service. For instance, the respective software can be packaged and shipped to the servers in the form of micro-VMs or containers, before the drones start their missions.

Let there be K edge servers, $s_k, 1 \leq k \leq K$, at different locations in the wider area where the drones operate. Each server s_k may have a different hardware platform hw_k with available computing resources res_k^{avl}. Also, each server is accessible via a dedicated local wireless network with bandwidth bw_k and communication range $range_k$. When drone d_m visits a point of interest $P_m[i]$ which is in range of server s_k, it can offload its computation $comp_m$ to it. Then, the computation delay for the drone is

$$compT(m,k) = procT(comp_m, hw_k) + \frac{data_m^{in} + data_m^{out}}{bw_k} \tag{2}$$

taking into account the time needed to perform the computation on the server and transfer the respective input and output over the network. For convenience, let $compT(m,0) = compT_m$ (the server identifier 0 means "no offloading").

Let $S_m[i]$ encode the offloading for d_m during its mission. More specifically, $S_m[i] = k$ if the drone shall offload $comp_m$ to s_k at $P_m[i]$, else $S_m[i] = 0$ if the drone shall perform the computation locally. Also, $S_m[i] = 0$ for $P_m[i] = dep_m$ as the drone does not perform any computation at the depot station.

Note that the drone may not be able to *immediately* offload its computation to $S_m[i]$ as soon as it completes sensing at $P_m[i]$. The reason is that server's resources may be used by other drones at that time. It is, however, possible for the drone to wait for the server to become available, and then proceed with the offloading process as usual. Let the extra waiting time for this be $W_m[i] \geq 0$. We let $W_m[i] = 0$ for all waypoints where $S_m[i] = 0$ and the drone does not use any server for offloading.

Based on the above, the time spent by the drone in order to perform the necessary sensing and computation at each point of interest $P_m[i] \in \mathcal{V}_m$, is

$$pT_m^i = senseT_m + W_m[i] + compT(m, S_m[i]) \tag{3}$$

Server Resource Usage Constraint. Let matrix U encode the planned usage of the server infrastructure, where $U[k][m][t] = 1$ if s_k allocates resources to run the service for $comp_m$ at time t, else $U[k][m][t] = 0$. Note that the total amount of resources allocated on a server due to offloading cannot exceed its overall resource capacity:

$$res_k^{avl} \geq \sum_{m=1}^{M} U[k][m][t] \times res_m^{req}, \forall k, t \tag{4}$$

We assume that the resources res_m^{req} associated with the service for performing $comp_m$, remain allocated only while the service is actually being used by a drone. More specifically, if d_m starts offloading its computation to s_k at time t_{start}, it will consume resources during the entire duration of the computation including the required input/output data transfers, $U[k][m][t] = 1, t_{start} \leq t \leq t_{start} + compT(m, k)$. Notably, we assume that an edge server cannot further offload a computation to another edge server or the cloud.

3.3 Energy Model

We assume battery-operated drones (however, our model can be equally applied to drones with an engine and a fuel tank). Let E_m^{max} be the maximum energy storage capacity of d_m's battery. Obviously, bigger values of E_m^{max} translate to larger autonomy, allowing the drone to operate for a longer amount of time before it needs to land.

Let the energy that is consumed by d_m to fly between two waypoints $P[i]$ and $P[i+1]$ be a linear function of flight time, $flyE_m^{i,i+1} = \beta_m \times flyT_m^{i,i+1}$. In the same spirit, the energy consumed by the drone to hover above $P[i]$ is a linear function of the time spent at that point in order to perform the required sensing and computing task, $pE_m^i = \gamma_m \times pT_m^i$. We consider the energy spent for onboard computation or the communication with a server to be negligible compared to the energy spent for hovering. Also, for polycopter drones flying at moderate speeds, we assume that $\beta_m \approx \gamma_m$, i.e., the energy spent for flying is comparable to the energy spent for hovering.

Safety Constraint. Let $remE_m^i$ denote the remaining energy of d_m at $P_m[i]$, after sensing and processing. Initially, $remE_m^1 = E_m^{max}$ as every drone starts from its depot with full batteries. The remaining energy at the next point in the drone's path is equal to the energy that was available at the previous point less the energy spent to fly to the next waypoint and the energy spent there (to hover while performing the required sensing and processing):

$$remE_m^{i+1} = remE_m^i - flyE_m^{i,i+1} - pE_m^{i+1} \tag{5}$$

Note that it is crucial to build flight paths and make offloading decisions so that the bellow equation holds:

$$remE_m^i > 0, \forall i, m \tag{6}$$

This constraint states that at no point during the execution of the mission should a drone deplete its batteries. Such an event would lead to an emergency landing, which is not only undesirable from a purely operational perspective but may also lead to material damages or even injuries, depending on the area of operation.

Depot Detours. To satisfy the above constraint, drones with limited autonomy or long missions may have to perform intermediate depot visits (detours) in order to switch batteries so that they can proceed with the rest of their mission. Let $depT_m$ denote the time it takes for d_m to switch batteries once it has landed at its depot dep_m. Taking this into account, we extend Eq. 3 to redefine the time spent at each point in the drone's path as

$$pT_m^i = \begin{cases} senseT_m + W_m[i] + compT(m, S_m[i]), & \text{if } P_m[i] \neq dep_m \\ depT_m, & \text{if } P_m[i] = dep_m \wedge 1 < i < \text{len}(P_m) \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

The first case captures the time spent at points of interest (all waypoints that do not represent a depot visit), as per Eq. 3. The second case captures the time spent at a depot to switch the drone's batteries before it can continue its mission. If, however, the drone is at its depot in the beginning and end of its path, the third case applies, setting the time spent at the depot to 0. This is because each drone starts its mission with fully charged batteries and the mission is considered to be completed the moment the drone lands at its depot for the last time (the battery switch time after the mission's completion does count in the actual mission time).

In the same vein, we re-define the energy spent at each point in the path as

$$pE_m^i = \begin{cases} \gamma_m \times pT_m^i, & \text{if } P_m[i] \neq dep_m \\ -E_m^{max} + remE_m^{i-1} - flyE_m^{i-1,i}, & \text{if } P_m[i] = dep_m \wedge remE_m^{i-1} - flyE_m^{i-1,i} > 0 \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

The first case applies to all points of interest, as already discussed above. The battery switching at the depot is captured by the second case, which sets the energy spent to a *negative* value so that the remaining energy of the drone $remE_m^i$ after a successful depot visit is equal to E_m^{max} as per Eq. 5. Note, however, that this artificial correction is allowed only if the drone has sufficient energy to reach the depot in the first place, as ensured by the second term of the condition for this case. Else, the third case of the above equation applies, which sets the energy spent at the depot to 0 so that $remE_m^i \leq 0$ as per Eq. 5, indicating that the drone will deplete its batteries before reaching the depot. As already stressed, this should never be the case in a properly planned mission.

3.4 Problem

Given the above, the time needed for d_m to complete its entire mission can be captured as:

$$mT_m = \sum_{i=1}^{\text{len}(P_m)-1} flyT_m^{i,i+1} + pT_m^{i+1} \tag{9}$$

Our goal is to exploit the available edge server infrastructure in order to reduce the mission time of all drones. More specifically, the problem we tackle in this work is the following:

Given drones $d_m, 1 \leq m \leq M$ with assigned points of interest $\mathcal{V}_m$ where they have to perform some sensing and data processing, and stationary servers $s_k, 1 \leq k \leq K$ located in the mission area, which can be used for offloading the drones' computations, produce paths P_m and offloading plans S_m that minimize mT_m (as per Eq. 9) *in a fair way for all drones*, while ensuring that the constraints of Eq. 4 and Eq. 6 are satisfied.

We capture the fairness objective by using as a reference the makespan of a default path P_m^{def} that does not involve any computation offloading. The corresponding mission time mT_m^{def} can be computed as per Eq. 9 for a special "empty" offloading schedule S^{def} where $S_m^{def}[i] = 0, 1 \leq i \leq \text{len}(P_m^{def})$. Based on this reference, we set as the optimization target

$$\text{maximize } min_{m=1}^{M}\left(\frac{mT_m^{def} - mT_m}{mT_m^{def}}\right) \tag{10}$$

In other words, we wish to maximize, across all drones, the relative reduction of their mission time, using the available server infrastructure for offloading. The rationale is for all concurrently running missions to benefit from the shared server resources in an even way.

We assume that the drone's maximum battery capacity E_m^{max} is sufficient so that d_m can move from its depot dep_m (starting with full batteries) to any point of interest $P_m[i]$ and back to dep_m even if the computation at that point is performed locally on the drone. This is to ensure that the problem always has a feasible solution irrespective of the points of interest assigned to each drone and the degree of contention among drones for the shared server resources.

4 Algorithm

Since the classic vehicle routing problem (VRP) is NP-hard, the above (more complex) problem is also NP-hard. Therefore, obtaining an exact solution for large instances of the problem would be too time consuming, even for an offline algorithm.

4.1 Overview

To tackle the above problem in reasonable time, we propose an energy-aware path planning and offload scheduling heuristic (EPPOS). The heuristic works in the spirit of a variable neighborhood search (VNS) algorithm [12], gradually improving the best solution found so far by exploring neighbouring solutions through random search. In our case, the solution consists of (a) the paths to be followed by the drones and (b) the schedule for offloading their computations to nearby edge servers. In turn, the offloading schedule is produced based on the expected computation time at each point of interest in the drone's path, and the so-called candidate selection order in which the drones are examined to find the next best possible offloading option. The latter is encoded as a sequence of drone identifiers, e.g. $[1, 2, 3.., 1, 2, 3..]$ means that the algorithm will first plan the next offloading option for drone d_1, then for d_2, then for d_3 etc. Note that the same identifier may appear more than once in this sequence, each such occurrence dictating when the algorithm will plan the next offload for that drone with respect to others.

4.2 Description

The high-level logic of the heuristic is given in Algorithm 1 while the main auxiliary functions are described in Algorithm 2. The algorithm takes as input the set of drones $\mathcal{D}$, an initial path P_m^{init} for each drone d_m (computed using an off-the-shelf TSP algorithm without considering energy constraints or computation offloading options), and the number of iterations of the optimization loop.

As a first step, it computes the default path P_m^{def} for each d_m by adjusting P_m^{init} so that it becomes fully safe assuming that all computations will be performed locally on the drone. More specifically, intermediate depot visits (detours) are inserted in the path to ensure that every hop can be performed based on the drone's battery level at each point, as per Eq. 6. Further, for each drone d_m, the expected time spent at each point of interest $pT_m^{exp}[i]$ is set equal to the sensing time $senseT_m$ plus the average computation time over all servers in range of $P_m[i]$ and the local computation at d_m (assuming that each option is equally probable and that d_m will not have to wait before starting the computation for that point of interest). As a last initialization step, the candidate selection order is set to a randomly chosen round-robin order.

Then the algorithm iteratively improves the paths and offloading schedule. Each iteration starts by resetting the path P_m of each d_m to P_m^{init} and adjusting these paths based on the expected times pT_m^{exp} spent at each point of interest, via function AdjustPaths(). More specifically, if the expected remaining energy of d_m does not suffice to reach all points of interest, its path P_m is considered unsafe and one or more intermediate depot visits (detours) are inserted at the proper points in order to switch batteries. This is done by finding the next waypoint in the path that cannot be reached because the drone will have exhausted its battery, and then checking the previous waypoints in the path (up to the last depot visit) to insert a detour between two waypoints so as to minimize the extra

Algorithm 1. Energy-aware path planning and offload scheduling (EPPOS)

```
function Main(𝒟, P^init, iterations)
    for each d_m ∈ 𝒟 do
        P_m^def ← calcSafePath(P_m^init)
        for each P_m[i] ≠ dep_m do
            pT_m^exp[i] ← senseT_m + avg(compT(m, k)), ∀s_k in range of P_m[i] and k = 0
        end for
    end for
    order ← rndRoundRobinOrder(𝒟)
    bestMinReduction ← −1

    for iterations do
        P ← AdjustPaths(𝒟, P^init, pT^exp)
        P, S, W, U, reductions ← ScheduleOffloading(𝒟, P, pT^exp, order)
        minReduction ← min(reductions)
        if minReduction > bestMinReduction then
            bestMinReduction ← minReduction
            bestP, bestS, bestW, bestU ← P, S, W, U
            bestOrder, bestpT^exp ← order, pT^exp
        end if
        order ← ShakeOrder(bestOrder, reductions)
        pT^exp ← AdaptExpTimes(bestpT^exp)
    end for
end function

function AdjustPaths(𝒟, P, pT^exp)
    for each d_m ∈ 𝒟 do
        if isUnsafe(d_m, P_m, pT_m^exp) then
            insertDepotDetours(P_m)
            trySwapReverseSubPaths(P_m)          ▷ if this reduces contention
        else
            tryReversePath(P_m)                  ▷ if this reduces contention
        end if
    end for
    return P
end function
```

Algorithm 2. Auxiliary functions

function ScheduleOffloading($\mathcal{D}, P, pT^{exp}, order$)
  for each $d_m \in \mathcal{D}$ **do**
    $pos_m, remE_m^1 \leftarrow 1, E_m^{max}$ ▷ start from $P_m[1] = dep_m$ with full batteries
    $S_m[i] = 0, W_m[i] = 0, 1 \leq i \leq \text{len}(P_m)$ ▷ no offloading / waiting times yet
    $U[k][m][t] \leftarrow 0, \forall k, t$ ▷ no server reservations yet
  end for
  while $order \neq \emptyset$ **do**
    $d_m \leftarrow$ pickNxtDrone($order$)
    $pos_m, P_m, k, t^{arr}, waitT \leftarrow$ fndNxtOffload($pos_m, P_m, pT^{exp}, S, W, U$)
    if $pos_m \neq len(P_m)$ **then** ▷ offload option found
      $S_m[pos_m], W_m[pos_m] \leftarrow k, waitT$
      $U[k][m][t] \leftarrow 1, \forall t, t^{arr} + waitT \leq t \leq t^{arr} + waitT + compT(m, k)$
    else ▷ arrived at the end of the path
      $order \leftarrow$ rmvAllEntries($order, d_m$)
      $reductions_m \leftarrow \frac{mT_m^{def} - mT_m}{mT_m^{def}}$ ▷ reduction as per Equations 9 and 10
    end if
  end while
  return $P, S, W, U, reductions$
end function

function ShakeOrder($order, reductions$)
  $moves \leftarrow$ pickRndRange($moves^{low}, moves^{high}$)
  $bestD \leftarrow$ sortDiscendingReductions($reductions, \mathcal{D}$)
  $worstD \leftarrow$ sortAscendingReductions($reductions, \mathcal{D}$)
  for m **from** 1 **to** $moves$ **do**
    $first\_occ \leftarrow$ pickRndRange($first\_occ^{low}, first\_occ^{high}$)
    $order \leftarrow$ moveFirstEntriesToTail($order, bestD[m], first\_occ$)
    $last\_occ \leftarrow$ pickRndRange($last\_occ^{low}, last\_occ^{high}$)
    $order \leftarrow$ moveLastEntriesToHead($order, worstD[m], last\_occ$)
  end for
  if pickRndBinary($prob_{swap}$) = $true$ **then**
    $swaps \leftarrow$ pickRndRange($swaps^{low}, swaps^{high}$)
    for $swaps$ **do**
      $m_1, m_2 \leftarrow$ pickRndPair($order$)
      $order \leftarrow$ swapEntries($order, m_1, m_2$)
    end for
  end if
  return $order$
end function

function AdaptExpTimes($bestpT^{exp}$)
  if pickRndBinary($prob_{pTcng}$) = $true$ **then**
    $pTchng \leftarrow pickRndRange[pTchng^{low}, pTchng^{high}]$
    $pT_m^{exp}[i] \leftarrow bestpT_m^{exp}[i] \times pTchng$ ▷ $\forall P_m[i] \neq dep_m$ in range of a server
  else
    $pT_m^{exp}[i] \leftarrow bestpT_m^{exp}[i]$ ▷ $\forall P_m[i] \neq dep_m$ in range of a server
  end if
  return pT^{exp}
end function

flight delay. As a result of such detours, P_m is effectively split in smaller subpaths separated by depot visits. In this case, the algorithm considers swapping the order of these subpaths as well as reversing the order in which the different points of interest are visited within each subpath, to reduce the expected contention for the available server resources. More precisely, the contention at each waypoint $P_m[i]$ is estimated by dividing the number of drones expected to arrive at points of interest covered by the servers that are in range of $P_m[i]$ (where the arrival time falls within the time window where offloading is still beneficial despite the potential waiting time due to server overload), with the number of servers that cover $P_m[i]$. High contention at a point of interest means that the drone has lower probability to successfully offload its computation to some server. To avoid performing an exhaustive search for this, we only consider a limited (up to a fixed maximum) number of swap/reversal samples, which are randomly selected in each iteration. Else, if P_m is safe (no depot detour is added), the algorithm simply considers a reversal of the entire path in order to reduce contention.

These path adjustments are illustrated in Fig. 1. Note that these adjustments are applied to the P_m only if they reduce the average contention along that path.

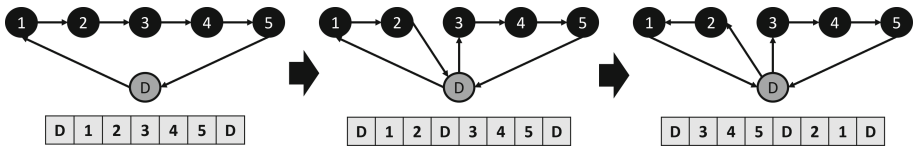


Fig. 1. Path adjustment example. The initial main path $[p_1, p_2, p_3, p_4, p_5]$ (left) is considered unsafe and as a consequence a depot detour is inserted between point of interest p_2 and p_3, resulting in two subpaths $[p_1, p_2]$ and $[p_3, p_4, p_5]$ (center). In addition, to minimize contention, the two resulting subpaths are swapped and the subpath $[p_1, p_2]$ is reversed and becomes $[p_2, p_1]$ (right).

After the paths are reset and adjusted, a new offloading schedule is produced via the ScheduleOffloading() function. This starts from a fresh state where all servers are unoccupied at all times and no computation offloading is planned for any drone. Then, drones are considered according to the candidate selection order, and for each one the next computation offload is planned. This is done via the $fndNxtOffload()$ function, which finds the next best safe offloading option in the drone's path (not shown for brevity). Notably, this function transparently handles the case where, as a side-effect of offloading decisions taken in previous steps, the actual time spent by d_m at some point of interest $P_m[i]$ may turn out to be different that what was expected, $pT_m[i] \neq pT_m^{exp}[i]$. If such individual or accumulated deviations render P_m infeasible, an intermediate depot detour is inserted in P_m in the drone's path to ensure the safety constraint as per Eq. 6 and the search for the next best offloading option is repeated. In any case, the function returns a possibly updated path and the offloading suggestion at the next possible point of interest, along with the respective server and waiting time

Table 1. Parameters guiding the random search of the EPPOS.

Parameter	Description
$moves^{low/high}$	Bounds for the random pick of the number of best and worst candidates to be moved to the tail and head of the candidate order/selection list, respectively.
$first\_occ^{low/high}$	Bounds for the random pick of the number of occurrences of the best candidate to be moved to the tail of the order list.
$last\_occ^{low/high}$	Bounds for the random pick of the number of occurrences of the worst candidate to be moved to the head of the order list.
$prob_{swap}$	Probability to swap entries in the order list.
$swaps^{low/high}$	Bounds for the random pick of the number of swaps to be performed in the order list.
$prob_{pTchng}$	Probability for changing $bestpT^{exp}$ to be used as input for the next iteration
$pTchng^{low/high}$	Bounds for the random pick of the change factor for $bestpT^{exp}$.

so that the server capacity constraint as per Eq. 4 is satisfied. If the returned point indeed corresponds to a point of interest, the offloading schedule is updated accordingly. Else, the returned point corresponds to the last point in the drone's path implying that no offloading opportunity was found for d_m, all occurrences of the drone are removed from the ordered candidate selection list.

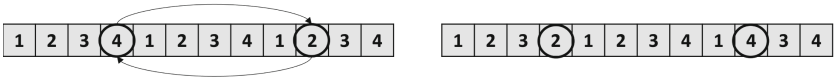
When the ScheduleOffloading() function returns, it is checked whether the newly produced paths and offloading schedule improves the optimization criterion vs the best solution found so far (achieves a higher worst relative reduction of the mission time across all drones). If so, the new paths and schedule, along with the corresponding candidate selection order and expected time spent by each drone at each point of interest, are stored as the best solution.

Finally, before entering the next iteration, the best candidate selection order and best expected times at each point are modified via the ShakeOrder() and AdaptExpTimes() function, respectively. These functions randomly change these crucial parameters in search for a better solution, and are briefly discussed in the following. Table 1 lists the parameters that guide this random search.

Function ShakeOrder() changes the best selection order found so far to build a new selection order in which the drones will be considered for offloading in the next iteration. This is done in two different ways. On the one hand, it picks *moves* drones that achieved the highest relative reduction of their mission times and for each one moves its $first\_occ$ first occurrences in the order list to the tail of the list. Also, it picks the same number of drones that achieved the lowest relative reduction of their mission times and for each one moves its *last_occ* last occurrences in the order list to the head of the list. On the other hand, with probability $prob_{swap}$, it swaps the order of two randomly chosen list entries. This is repeated for *swaps* times. Figure 2 illustrates simplified scenarios for those operations. The rationale of the first transformation is to strike a

better balance by promoting the drones with the worst mission time improvement so that they are considered earlier for offloading in the next iteration, while demoting the drones with the best improvement. The second transformation is a simple random mutation.

(a) Move the first occurrence of the drone with the highest relative reduction of the mission time, let d_2, to the tail of the list, and move the last occurrence of the drone with the lowest relative reduction, let d_3, to the head of the list.



(b) Throw a binary dice which gives *true* with probability $prob_{swap}$. If *true*, pick two random occurrences in the list, e.g., the first occurrence of d_4 and the last occurrence of d_2, and swap them.

Fig. 2. Shake operations (mutations) on the order/candidate selection list (for $moves = 1$, $first\_occ = 1$, $last\_occ = 1$, $swaps = 1$).

Function AdaptExpTimes() adapts the expected time that will be spent by the drones at each point of interest, based on the values that were used to find the best solution so far. More specifically, with probability $prob_{pTchng}$, the expected times at each point of interest for the next iterations are set to $bestpT_m^{exp}[i]$ adapted by a factor $pTchng$, else they are set equal to $bestpT_m^{exp}[i]$. The underlying rationale for this adaptation is to compensate for the fact that these estimates are based on rough approximations for the time that will be spent at each point of interest to perform the computation. Note that all the above variables ($moves$, $first\_occ$, $last\_occ$, $swaps$, $pTchng$) are randomly picked from respective intervals that are specified via the bounds given in Table 1.

4.3 Complexity

The runtime complexity of the EPPOS heuristic is determined by the number of iterations and the complexity of the procedures that are executed in each iteration, briefly discussed below. For brevity we let $V = \sum_{m=1}^{M} |\mathcal{V}_m|$ denote the total number of visits to points of interest to be performed by the drones. Also, let $D = \frac{\sum_{m=1}^{M} max_{s_k \text{ in range of } \mathcal{V}_m}(compT_m - compT(m,k))}{M}$ be the maximum difference between the local computation time of a drone and the computation time at the fastest server in range of a point of interest, averaged over all drones. Note that

D corresponds to the average number of checks needed to determine whether a drone can offload its computation to a server at an acceptable waiting time so that this is still beneficial compared to performing the computation locally.

The complexity of the path adjustment procedure (AdjustPaths function) is $O(V) + O(M \times V \times D \times K)$, for the insertion of depot detours and the consideration of path reversal/swaps, respectively. In the former case, the path of each drone needs to be traversed linearly (with limited backwards checks that do not increase the complexity). In the latter case, for each drone, a fixed number of subpath reversal/swap combinations are checked for contention against the path of every other drone. In turn, the contention check at each point of interest is done for all servers in range and for the time slots where computation offloading remains beneficial. Resource allocation (ScheduleOffloading function) has a complexity of $O(V \times D \times K)$ because for each point of interest in the path of each drone, all servers in range are checked for availability for the time slots where offloading is still beneficial, in order to decide whether, where and when to offload the computation of the drone. The random adaptation of the expected time spent by each drone at each point of interest (AdaptExpTimes function) has a complexity of $O(V)$. Finally, the complexity of the random rearrangement of the candidate selection list (ShakeOrder function) is $O(V)$ as the total number of performed mutations (regulated through the random variables $moves$, $first_{occ}$, $last_{occ}$ and $swaps$) cannot be larger than V.

It follows that the aggregate complexity of an iteration is $O(M \times V \times D \times K)$. Even though this is not negligible, it is relatively lightweight compared to an exhaustive check of all possible drone path and offloading combinations, allowing the algorithm to scale for larger instances of the problem.

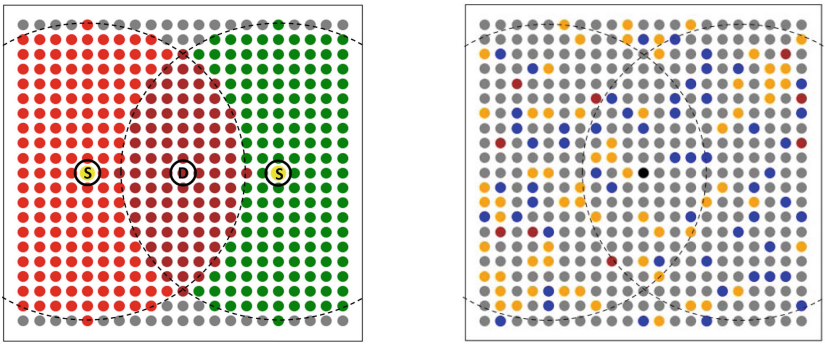
5 Evaluation

We evaluate the EPPOS algorithm through simulation experiments. The goal of our evaluation is to demonstrate the performance, robustness and fairness of EPPOS for a wide range of scenarios, in particular regarding the battery switching time at the depot station and the operational autonomy of the drone.

Although we have the capability to experiment with real drones, such tests in the field come with numerous limitations. In particular, various flight restrictions but also changing weather conditions make it very hard, if not practically impossible, to conduct a large number of experiments with long-running missions over a wider area so that the different results can be compared with each other in fair way. This is why we resort to simulations. However, for both the drone and the server we use realistic parameter settings, based on the real hardware platforms we use in our field experiments.

5.1 Topology and Missions

The missions are conducted inside a rectangular area, with 441 different locations arranged in a 21×21 grid, as shown in Fig. 3a. The nodes of the grid serve as potential locations of interest of a mission. Neighboring vertical and horizontal nodes are 20 m apart. The depot stations of all drones are located at the center of the grid.



(a) Topology of mission area. (b) Two randomly generated missions.

Fig. 3. Mission area for the experiments.

We assume two servers in the area of operation, near the yellow nodes marked with an S. Each server has a WiFi interface with a range of 200 m at constant throughput of 52 Mbps (this was measured via the NS3 [15] for TCP/IP communication over high gain antennas). The dashed lines and coloured areas mark the regions from where each server can be used for offloading: red for the server on the left; green for the server on the right; brown points are covered by both servers. The gray nodes are out of range for both servers.

We perform our experiments for randomly generated missions in the above topology, by randomly picking for each drone from 60 up to 80 points of interest (each drone is assigned a different random mission). Two indicative randomly generated missions are shown in Fig. 3b. The blue and orange nodes denote the points of interest for the first and second mission, respectively, while the brown nodes are common for both missions. These points of interest are fed into a typical TSP algorithm to produce the initial paths (P^{init}) for the drones, which, in turn, are handed over to the EPPOS algorithm in order to produce the optimized drone paths and offloading plan to the shared edge infrastructure.

5.2 Drone and Server Model Parameters

In the experiments presented in this paper, we assume drones with the same flight characteristics and onboard computing platform. As an indicative embedded

computing platform, featured in the drone we use for our field experiments, we consider an RPi 4 model B with 2 GB of RAM running a headless Ubuntu server 20.04. The RPi also has a WiFi interface through which it can communicate with the edge servers.

At each point of interest, the drone takes a picture using its onboard camera and processes the image in order to detect certain objects of interest. The sensing delay (image capture) is set to 1 s (as measured on the RPi) while the image captured is roughly 465 KB. Object detection is done using YOLO [14]. The local computation delay on the RPi was measured at roughly 10 s.

For the horizontal movement of the drone, we assume steady acceleration 0.8 m/s^2 when departing from a waypoint until the drone reaches the operational speed of 4 m/s, and steady deceleration 1.6 m/s^2 as the drone approaches and stops at the next waypoint. So, for example, the distance of 20 m between two neighbouring points in the above topology (Fig. 3a), is covered in about 9 s. Also, we assume that the drone needs 5 s for a vertical take-off from the depot to the operational flight altitude of 10 m, and 20 s to perform vertical landing at the depot. These times were confirmed using our drone running the ArduPilot [1] autopilot stack onboard as well as via the ArduPilot software-in-the-loop configuration, which produced similar results.

The autonomy of the drone is the total amount of time it can remain in the air based on its battery capacity, including the time needed to take-off from the depot, fly from one point of interest to the next, hover over a point of interest to perform the required sensing and data processing, and to land at the depot. As already discussed in the previous section, drones always leave the depot with their batteries fully charged. We assume that drones consume the same amount of energy when hovering, flying, taking-off and landing. This assumption is quite realistic for small drones given that the above flight model includes moderate acceleration/deceleration values.

As an edge server platform we use a laptop with an i7-8550U CPU running a VM with 4 cores and 4 GB of RAM. The object detection service used for offloading runs inside the VM as a container using Docker [11]. The total computation delay including container loading and the time that is needed to transmit the request with the image and the detection results over WiFi, is measured at about 2 s. The number of drones that can be served concurrently by each such server is set to 1, because we observed a significant increase in the processing time when two or more services run concurrently in the VM.

5.3 Settings for EPPOS Parameters

The settings for the parameters of the random search heuristic of EPPOS are given in Table 2. We arrived at those values empirically, after a number of exploratory runs. With these settings, the algorithm converges quickly to a good solution. More specifically, in practically all the exploratory runs, convergence was reached long before 400 iterations – additional iterations had a negligible effect on the quality of the solution yielding marginal or no improvement. Thus, in the experiments presented here, we set the *iterations* parameter to 400.

Table 2. Settings for the parameters guiding the random search of EPPOS.

Parameter	value
$moves^{low}$, $moves^{high}$	1, $\frac{M}{3}$
$first\_occ^{low}$, $first\_occ^{high}$	1, 5
$last\_occ^{low}$, $last\_occ^{high}$	1, 5
$prob_{swap}$	0.1
$swaps^{low}$, $swaps^{high}$	$\frac{M}{2}$, M
$prob_{pTchng}$	0.6
$pTchng^{low}$, $pTchng^{high}$	0.7, 1.3

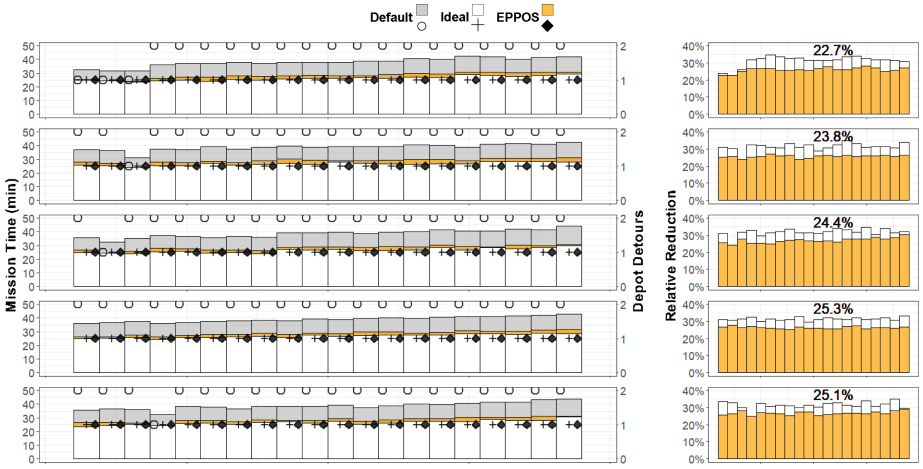
We note that the time that is needed to perform these iterations is acceptable for an offline algorithm, roughly 1 min. This time was measured on a commodity laptop running Windows10, with i7-8550U CPU and 8 GB of RAM, with an implementation of the EPPOS algorithm in Python3.

5.4 Presentation of Results

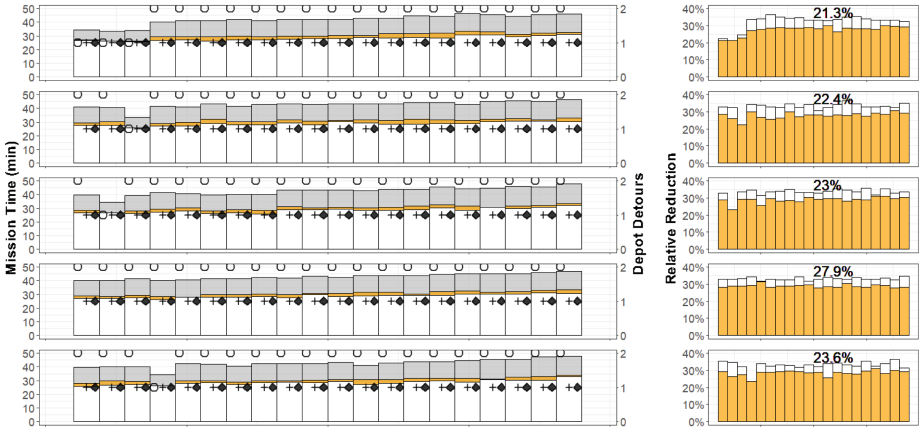
In the sequel, we report the results of experiments that are performed for 20 drones operating concurrently in the mission area, where we vary (i) the battery switching time at the depot and (ii) the autonomy of the drones.

As a reference for the results produced by EPPOS, we use a deterministic algorithm that optimizes the initial paths (P^{init}) under the assumption that each drone can immediately offload its computation, with zero waiting time, to one of the servers that are in range of the respective points of interest. This algorithm works in a similar way to the part of the AdjustPaths() function (see Algorithm 1) where intermediate depot detours are added to make the path safe, without the path reversal logic. Note, however, that since the algorithm assumes perfect offloading, the result corresponds to a *potentially infeasible* lower bound for the mission time (and upper bound for the maximum relative reduction vs the default mission time), for the ideal case where there is no contention between the drones that operate concurrently in the mission area.

For each experiment scenario, we report the outcome of 5 different sets of randomly generated missions (every set consists of 20 missions, one mission per drone). Given that the EPPOS algorithm is itself randomized, we run it five times for each mission set and report the median. The results are presented via two barplots placed next to each other (one bar per drone, one row per mission set). In the left plot, the bars show the mission time achieved by EPPOS vs the default and ideal mission time for each drone (left Y-axis). We also indicate the number of depot detours that are introduced for each drone (right Y-axis). In the right plot, we show the relative reduction for EPPOS as well as for the ideal case assuming no contention vs the default mission time. The percentage on top of the plot indicates the worst relative reduction that is achieved by EPPOS over all drones (the optimization objective).



(a) Battery switching delay is 3 minutes.



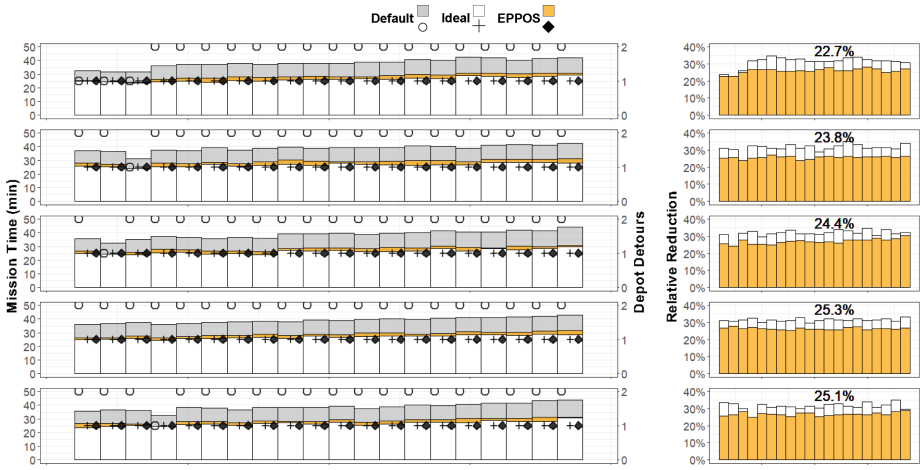
(b) Battery switching delay is 5 minutes.

Fig. 4. Different battery switching delays (autonomy is 15 min).

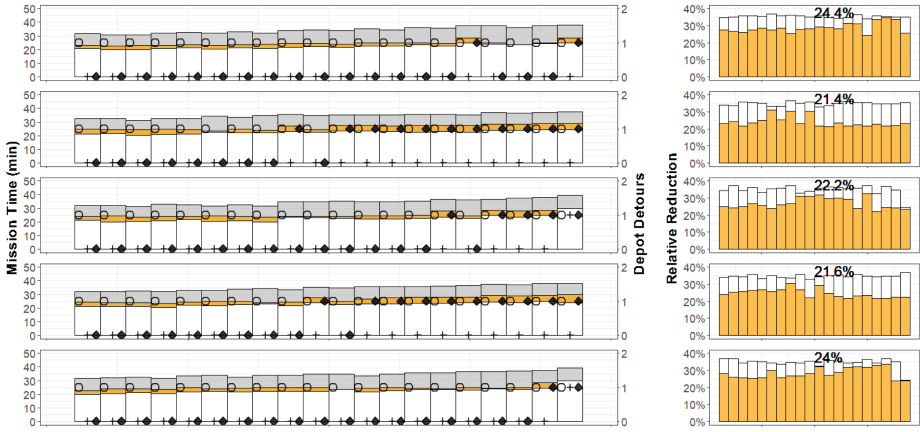
5.5 Experiments for Different Battery Switching Delays

In a first set of experiments we study the performance of the EPPOS algorithm for battery switching delays of 3 min and 5 min, for an autonomy of 15 min. The results are shown in Fig. 4. Averaged over all five randomly generated path sets, the worst relative reduction of the mission time across all drones is 24.3% and 23.6% for a battery switching delay of 3 and 5 min, respectively. Compared to the ideal case of zero contention, the results of EPPOS are worse by merely 0.89% and 0.75%.

The difference between EPPOS and the ideal case becomes less significant for an increasing battery switching delay. The reason is twofold. Firstly, larger



(a) Drone autonomy is 15 minutes.



(b) Drone autonomy is 25 minutes.

Fig. 5. Different degrees of autonomy (battery switching delay is 3 min).

battery switching delays reduce contention as every drone that makes a depot detour spends more time at the depot and as a result stays out of competition for the shared edge resources for a longer period. Secondly, when the battery switching delay is large and a depot detour is unavoidable in order for a drone to safely complete its mission, the time that can be saved thanks to offloading becomes less significant for the total mission time.

5.6 Experiments for Different Degrees of Autonomy

In a second set of experiments, we keep the battery switching delay to 3 min and vary the autonomy of the drones. In addition to the autonomy of 15 min,

we investigate scenarios with an autonomy of 25 min. The results are shown in Fig. 5. Note that Fig. 5a is identical to Fig. 4a; it is shown again here to enable a direct visual comparison with the larger autonomy scenario.

Averaged over all five randomly generated path sets, EPPOS achieves a worst relative reduction of the mission time across all drones, of 24.3% and 22.7% for an autonomy of 15 and 25 min. The result is 0.89% and respectively 7% worse than the ideal case. Note that the difference of EPPOS vs the ideal increases for the scenario with a larger autonomy. The reason is that the ideal mission planning algorithm assumes zero contention and can fully exploit the larger autonomy to complete the mission with fewer depot detours. However this does not mean that theses savings are entirely realistic given that the actual contention between drones is non negligible. This, in turn, may lead to the loss of some offloading opportunities and force EPPOS to introduce additional depot detours in order for the drones to be able to complete their mission. It is also possible that the algorithm fails to escape a local optimum in the solution space.

An important observation, which holds for the previous experiments, is that EPPOS manages to produce balanced plans where all drones achieve a similar reduction of their default mission time, which is particularly valuable in light of fairness. Note that, as a side-effect of the optimization criterion, EPPOS tends to maximize the reduction for the drones that have lower improvement potential (shortest white bars) before further improving the mission times of others.

6 Conclusion

We have formulated the joint problem of path planning and edge resource allocation with the objective to maximize the worst relative reduction of the drones' mission times vs a default mission with no offloading, while satisfying energy and resource constraints. To address the problem, we have proposed an algorithm following a randomized neighborhood search. Overall, the algorithm achieves good results, also when compared to the ideal case (assuming no contention for the shared server resources) and improves the mission times evenly among drones.

As a possible direction for future work, it could be interesting to transform the problem into an equivalent formulation of a more abstract optimization problem, such as an integer programming (IP) problem, and compare the results of the proposed algorithm with standard solutions that have been proposed for the latter. The problem could also be extended to consider offloading among edge servers via a backbone network as well as between the drone and the cloud via a direct 4/5G mobile network link.

References

1. Ardupilot web site. https://ardupilot.org
2. Al-Hilo, A., Samir, M., Assi, C., Sharafeddine, S., Ebrahimi, D.: UAV-assisted content delivery in intelligent transportation systems-joint trajectory planning and cache management. IEEE Trans. Intell. Transp. Syst. **22**(8), 5155–5167 (2020)

3. Asim, M., Mashwani, W.K., Belhaouari, S.B., Hassan, S.: A novel genetic trajectory planning algorithm with variable population size for multi-UAV-assisted mobile edge computing system. IEEE Access **9**, 125569–125579 (2021)
4. Atencia, C.R., Del Ser, J., Camacho, D.: Weighted strategies to guide a multi-objective evolutionary algorithm for multi-UAV mission planning. Swarm Evol. Comput. **44**, 480–495 (2019)
5. Chemodanov, D., Qu, C., Opeoluwa, O., Wang, S., Calyam, P.: Policy-based function-centric computation offloading for real-time drone video analytics. In: 2019 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), pp. 1–6. IEEE (2019)
6. Chen, J., Chen, S., Luo, S., Wang, Q., Cao, B., Li, X.: An intelligent task offloading algorithm (iTOA) for UAV edge computing network. Digit. Commun. Netw. **6**(4), 433–443 (2020)
7. Kasidakis, T., Polychronis, G., Koutsoubelias, M., Lalis, S.: Reducing the mission time of drone applications through location-aware edge computing. In: IEEE International Conference on Fog and Edge Computing (ICFEC), pp. 45–52 (2021)
8. Khan, A.W., Abdullah, A.H., Anisi, M.H., Bangash, J.I.: A comprehensive study of data collection schemes using mobile sinks in wireless sensor networks. Sensors **14**(2), 2510–2548 (2014)
9. Kim, B., Min, H., Heo, J., Jung, J.: Dynamic computation offloading scheme for drone-based surveillance systems. Sensors **18**(9), 2982 (2018)
10. Liu, Y.: An optimization-driven dynamic vehicle routing algorithm for on-demand meal delivery using drones. Comput. Oper. Res. **111**, 1–20 (2019)
11. Merkel, D., et al.: Docker: lightweight Linux containers for consistent development and deployment. Linux J. **2014**(239), 2 (2014)
12. Mladenović, N., Hansen, P.: Variable neighborhood search. Comput. Oper. Res. **24**(11), 1097–1100 (1997)
13. Rabta, B., Wankmüller, C., Reiner, G.: A drone fleet model for last mile distribution in disaster relief operations. Int. J. Disaster Risk Reduction **28**, 107–112 (2018)
14. Redmon, J., Farhadi, A.: YOLOv3: an incremental improvement. arXiv preprint arXiv:1804.02767 (2018)
15. Riley, G.F., Henderson, T.R.: The *ns*-3 network simulator. In: Wehrle, K., Güneş, M., Gross, J. (eds.) Modeling and Tools for Network Simulation, pp. 15–34. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12331-3_2
16. Sabor, N., Sasaki, S., Abo-Zahhad, M., Ahmed, S.M.: A comprehensive survey on hierarchical-based routing protocols for mobile wireless sensor networks: review, taxonomy, and future directions. Wirel. Commun. Mob. Comput. **2017**, 23, Article ID 2818542 (2017). https://doi.org/10.1155/2017/2818542
17. Stavrinides, G.L., Karatza, H.D.: A hybrid approach to scheduling real-time IoT workflows in fog and cloud environments. Multimed. Tools Appl. **78**(17), 24639–24655 (2019). https://doi.org/10.1007/s11042-018-7051-9
18. Xiong, J., Guo, H., Liu, J.: Task offloading in UAV-aided edge computing: bit allocation and trajectory optimization. IEEE Commun. Lett. **23**(3), 538–541 (2019)
19. Zhan, C., Hu, H., Sui, X., Liu, Z., Wang, J., Wang, H.: Joint resource allocation and 3D aerial trajectory design for video streaming in UAV communication systems. IEEE Trans. Circ. Syst. Video Technol. **31**(8), 3227–3241 (2020)
20. Zhang, J., et al.: Stochastic computation offloading and trajectory scheduling for UAV-assisted mobile edge computing. IEEE Internet Things J. **6**(2), 3688–3699 (2018)