INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

WILEY

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. 0 (2024) 1–24
DOI: 10.1111/itor.13457

The multidepot drone general routing problem with duration and capacity constraints

Teresa Corberán[a], Isaac Plana[b,*], José M. Sanchis[c] and Paula Segura[c]

[a] *Departament d'Estadística i Investigació Operativa, Universitat de València, Avda. Dr. Moliner 50, Burjassot 46100, Spain*
[b] *Departamento de Matemáticas para la Economía y la Empresa, Universitat de València, Avda. Tarongers s/n, València 46022, Spain*
[c] *Departamento de Matemática Aplicada, Universitat Politècnica de València, Camino de Vera s/n, València 46022, Spain*
E-mail: tecorfa@alumni.uv.es[Corberán]; isaac.plana@uv.es[Plana]; jmsanchis@mat.upv.es[Sanchis]; psegmar@upvnet.upv.es[Segura]

Received 24 May 2023; received in revised form 18 January 2024; accepted 13 March 2024

Abstract

This paper studies the multidepot drone general routing problem with duration and capacity constraints (MDdGRP), an extension of the classical general routing problem with several depots. A fleet of drones with limited flight time and payload, each one located in a different depot, must jointly perform the service. In this continuous optimization problem, a set of lines of a network has to be serviced and a set of nodes has to be visited to deliver a given commodity. The aim is to find a set of drone routes performing the complete service with the shortest total duration. Aerial drones can enter a line at any point, service a portion of that line, and exit through another point, without following the lines of the network. While this flexibility allows for potentially better solutions, it also adds complexity to the problem. To deal with this, we digitize the MDdGRP instances by approximating each line with a polygonal chain containing a finite number of points that can be used to enter and exit each line. Thus, an instance of a discrete multidepot general routing problem (MDGRP) is obtained. This paper proposes an integer formulation for the MDGRP, some families of valid inequalities, and a branch-and-cut algorithm based on the strengthened formulation. Two matheuristic algorithms are presented, focused on sequentially selecting (and adding) some promising intermediate points to enter and exit a line. Instances involving up to six depots, 22 delivery points, and 88 original lines have been generated to test the performance of the methods proposed.

Keywords: drones; general routing problem; multi-depot; length constraints; capacity constraints; matheuristic; branch-and-cut

1. Introduction

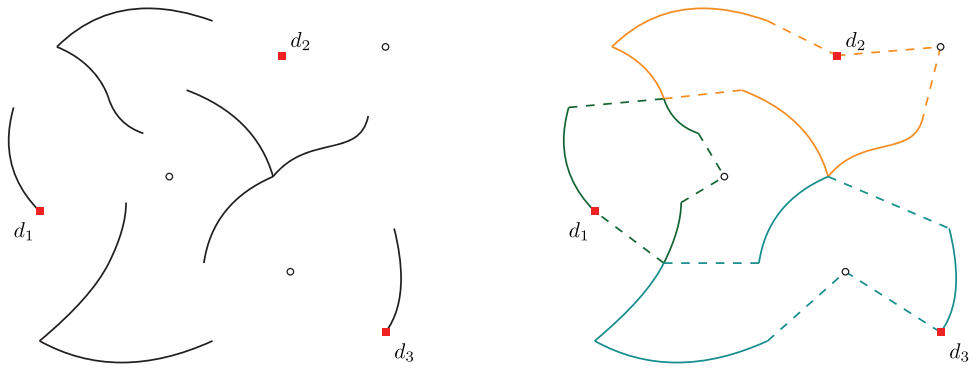
Routing problems have been usually categorized into two families: node routing problems, in which a service (e.g., delivering a package) has to be performed at certain points of a network, and arc

routing problems, where the service (e.g., street cleaning) is carried out by traversing some links of the network. The routing problem in which the services must be performed both at some arcs and vertices of the network is called the general routing problem (GRP), introduced by Orloff (1974). Given an undirected graph, the GRP consists of finding the minimum-cost route (closed walk) that visits a subset of vertices, called required vertices, and traverses a subset of edges, called required edges. The GRP was proved to be NP-hard in Lenstra and Rinnooy Kan (1976) and, therefore, exact algorithms are only useful for GRP instances of moderate size. Since then, the GRP has been studied in undirected, directed, mixed, and windy graphs (see, e.g., Corberán et al., 2015). Perhaps the best exact algorithm for the GRP is the one proposed in Corberán et al. (2007), a branch-and-cut algorithm incorporating several families of valid inequalities that is capable of solving to optimality GRP instances with up to 1000 vertices and more than 3000 edges. A lower bound for the GRP is proposed in Bach et al. (2013).

A natural extension of the GRP appears when we consider a limitation on the amount of the service a single vehicle can perform, thus obtaining the capacitated GRP (CGRP), which is a multivehicle routing problem (Pandi and Muralidharan, 1995; Prins and Bouchenoua, 2005; Bosco et al., 2013). Archetti et al. (2017) introduce the profitable CGRP, in which there is a profit associated with the service of both some edges and nodes. In Ahabchane et al. (2020), the mixed CGRP with time-dependent demands is considered, while Salazar-Aguilar et al. (2013) present a "road-marking" application, where both nodes and arcs require service and can be modeled as a synchronized GRP. Hasle (2014) argues that the mixed CGRP is an adequate model for routing applications where, although demand is fundamentally point based, groups of vertices can be represented as required arcs or edges by aggregating their demands. Oruc and Kara (2018) propose an extension of the GRP applied to the early assessment of damage in populated areas after a disaster by teams equipped with drones and motorcycles.

In problems with several vehicles, it may make sense to consider several depots from which the vehicles start their routes. Node routing problems with several depots have been extensively studied in the literature (see, e.g., the survey of Montoya-Torres et al., 2015). Regarding arc routing problems with several depots, however, the literature is scarce, and, as far as we know, there are no previous works on the GRP with multiple depots.

Most of the works mentioned above consider traditional ground vehicles. However, the use of drones for delivery or inspection tasks is becoming more common each day, since they offer a number of advantages compared to traditional means of transportation, being able to reach places that are difficult to access or dangerous for people or conventional vehicles. Routing problems with drones have some characteristics that set them apart from those with ground vehicles. Drones can travel in a straight line between any two points, without following the links of the network as happens in classical routing problems. If the service to be performed consists of traversing some lines, the drones can enter one of these lines at any point (even in the middle of it), traverse and service part of it, leave the line at another point, and then travel directly to another location to perform another service. This may allow shorter routes than those obtained with traditional vehicles but turns the routing problem into a continuous optimization one that is more difficult to solve. Recent advances in drone technology have led to the appearance of models capable of performing several tasks, such as making deliveries and mapping surfaces, in the same flight. These types of problems can be modeled as GRPs.



(a) MDdGRP instance (b) MDdGRP solution

Fig. 1. MDdGRP instance and a feasible solution.

The problem studied in this paper is the multidepot drone general routing problem with duration and capacity constraints (MDdGRP). In the MDdGRP, there is a set of lines that have to be traversed, each one with an associated traversal time, and a set of locations where a given demand must be delivered, also with a given delivery time. To perform these tasks, there is a fleet of identical drones. Each drone starts and ends its route at a given point, a depot, that can be different for each drone. The drones have both a maximum payload Q that they can carry and a maximum flight time L. The objective of the MDdGRP is to find a route for each drone, each one of them starting and ending at its depot, in such a way that they jointly perform all the tasks with minimum total duration (or cost). The duration of each route must not exceed L, and the capacity Q of the drones must not be exceeded. In this work, we will assume that the maximum flight time L and the capacity Q are enough to traverse all the lines and deliver all the demand. If this was not the case in a real-life application, we could consider the possibility of allowing the drones to come back to the depot, change their battery or recharge them, load more items to be delivered, and continue flying. This can be modeled in our problem by adding more copies of the depot and one new drone for each of these copies. These new drones would represent consecutive flights of the same drone, thus allowing us to obtain solutions for such instances, although at the expense of increasing their difficulty.

To illustrate the problem we address here, Fig. 1a shows an MDdGRP instance with three depots, d_1, d_2, and d_3 (represented by red squares), a set of three required locations that have to be visited (represented by circles), and a set of lines requiring a service (e.g., inspection or surveillance). Let us suppose that each delivery has demand 1 and each drone has capacity $Q = 1$, that is, each drone can perform one delivery. A feasible solution for this MDdGRP instance is given in Fig. 1b, where each route is depicted in a different color. Each drone route starts and ends at the same depot and satisfies the duration and capacity constraints. In this example, the service of some of the lines is shared by two different routes.

There is a large number of scientific articles in the literature regarding routing problems with drones. Regarding node routing problems, Murray and Chu (2015) introduced the flying sidekick traveling salesman problem, which considers the cooperation of one truck and several drones to provide service to a set of customers, and several generalizations of this problem have been

addressed since then. Wang and Sheu (2019) presented the vehicle routing problem with drones, where there is a fleet of trucks equipped with drones that can be dispatched from and picked up at any node of the network, and Di Puglia Pugliese and Guerriero (2017) extended it to consider time windows for each customer. Kim and Moon (2024) introduced a new problem where the drones start at different depots and the location of these depots has to be determined. The surveys of Macrina et al. (2020) and Rojas Viloria et al. (2021) provided an extensive overview of recent works on node routing problems with drones.

Drone arc routing problems were studied for the first time in Campbell et al. (2018), where for each line a finite number of points is selected so that drones can enter and leave the line only at these points. In other words, each original line is now divided into small edges that a drone must traverse completely or not. This same approach has been used in Campbell et al. (2021, 2022) to solve the length-constrained K-drones rural postman problem, in which a fleet of drones with limited autonomy must traverse a set of lines. Other works have dealt with drones in arc routing problems. Chow (2016) studies a stochastic arc-inventory routing problem applied to traffic monitoring. He proposes an approximate dynamic programming algorithm based on least squares Monte Carlo simulation to derive its stochastic dynamic policy. In Li et al. (2018), the authors propose a mixed integer programming formulation combining the capacitated arc routing problem with the inventory routing problem for solving a drone scheduling problem in traffic monitoring. The combination of other types of vehicles with drones in the context of arc routing problems has also been studied in the literature. Luo et al. (2019) model the problem as a double-layer arc routing problem, present a mixed integer linear programming formulation, and propose heuristic algorithms for its solution. Amorosi et al. (2021, 2024) describe problems in which the drone is carried by a mothership that can move freely in a two-dimensional space, a connected piecewise linear polygonal chain, or a general graph. They propose mixed integer second-order cone nonlinear programming formulations for all these situations and develop a matheuristic algorithm.

Few works have dealt with drones that have to perform services at arcs and vertices of the network in the same flight. Campbell et al. (2023) detailed a real-life application, motivated by a UNICEF project in Malawi, in which this type of drone, called "multipurpose," would be used to transport packages (e.g., medical aid) and, at the same time, take images of areas that have suffered some kind of natural catastrophe, such as floods or earthquakes. As in our work, that problem is modeled as a GRP, but drones are restricted to starting and ending at a single depot. The authors present a formulation for the problem and propose branch-and-cut and matheuristic algorithms.

As some of the drone routing problems mentioned above, the MDdGRP is a continuous optimization problem, since drones can enter and leave each line at any of its points. However, if the lines that have to be traversed are approximated by polygonal chains and we allow drones to enter and exit each line only at the vertices of these polygonal chains, we obtain a discrete problem, the multidepot general routing problem (MDGRP). If the number of points of the polygonal chains is large enough, the optimal solution of the MDGRP will be very similar to the optimal solution of the MDdGRP. Therefore, we will study the MDGRP as a means to obtain solutions for the MDdGRP.

The main contributions of this paper are the following. We introduce a new problem, the MDdGRP, based on the GRP with drones and considering capacity constraints and multiple depots. Since this is a continuous optimization problem, we formulate a discrete relaxation of the MDdGRP, the MDGRP, and propose several families of valid inequalities that strengthen this formulation. Using these inequalities, we develop a branch-and-cut algorithm to solve the MDGRP.

We also propose two matheuristic algorithms to obtain quality solutions for the MDdGRP. A new set of benchmark instances for the MDdGRP has been generated, and extensive computational experiments have been conducted to assess the performance of the proposed methods.

The remainder of this paper is structured as follows. In Section 2, we propose a formulation for MDGRP and present several valid inequalities, while a branch-and-cut algorithm for its exact solution is described in Section 3. Section 4 is devoted to the description of two matheuristic algorithms for the MDdGRP. The computational experiments are reported in Section 5, and some conclusions and future research lines are discussed in Section 6.

2. The multidepot general routing problem

The MDGRP is defined on an undirected graph $G = (V, E)$, with vertex set V and edge set $E = E_R \cup E_{NR}$. The set of required edges E_R corresponds to those that have to be traversed, and the remaining edges in E, the set of nonrequired edges E_{NR}, are used for deadheading. The set V contains the vertices incident with the edges in E_R, denoted V_I, the set of depots $\mathcal{D}$, and a set of required vertices with positive demand V_R that must be visited. Therefore, $G = (\mathcal{D} \cup V_R \cup V_I, E_R \cup E_{NR})$. We assume that the required vertices cannot be depots or incident with required edges, that is, $V_R \cap (\mathcal{D} \cup V_I) = \emptyset$, although it is possible that set V_I contains some depot $d \in \mathcal{D}$. There is a traversal (and service) cost $c_e^s \geq 0$ associated with each required edge $e \in E_R$, and a deadheading cost $c_e \geq 0$ associated with each nonrequired edge $e \in E_{NR}$. Each node $i \in V_R$ has an associated positive integer demand $d_i > 0$ and a service cost $c_i \geq 0$. The cost (duration) of a route is the sum of the costs of the edges traversed and the vertices serviced in the route. We consider a fleet of vehicles, each one associated with a different depot, with a limit L on the duration of their routes and another limit Q on the maximum load that they can carry. An MDGRP *solution* is a set of $D = |\mathcal{D}|$ routes, each one starting and ending at the corresponding depot, that jointly traverses all the required edges and visits all the required vertices, satisfying the capacity and duration limitations. The goal is to find an MDGRP solution with minimum total cost.

Since the MDGRP instances we want to solve in this work have originated from the discretization of MDdGRP instances, here the set of required edges E_R contains the segments of the polygonal chains into which the original lines have been transformed. The cost of each line is distributed among the segments proportionally with respect to the length of each segment. Moreover, since drones can fly directly between any pair of points, the set of nonrequired edges forms a complete graph.

The formulation we propose for the MDGRP considers a binary variable x_e^d for each edge $e \in E$ and each depot $d \in \mathcal{D}$, and a binary variable y_e^d for each nonrequired edge $e \in E_{NR}$ and for each depot $d \in \mathcal{D}$. Variable x_e^d takes the value 1 if the required edge e is traversed (and serviced) in the route of depot d and 0 otherwise, while variables x_e^d and y_e^d take the value 1 if the nonrequired edge e is traversed once or twice, respectively, in the route starting and ending at depot d, and 0 otherwise. Note that variables x_e^d and y_e^d represent the first and second traversals of nonrequired edge e in route d. Additionally, a binary variable z_i^d is introduced for each vertex $i \in V_R$ and each depot $d \in \mathcal{D}$, taking the value 1 if the vertex i is serviced in the route of depot d and 0 otherwise.

We will use the following notation. Given two subsets of vertices $S, S' \subseteq V$, $(S : S')$ denotes the edge set with one endpoint in S and the other in S'. Given a subset $S \subseteq V$, let us denote $\delta(S) = (S :$

$V \setminus S$) and let $E(S) = \{e = (i, j) \in E : i, j \in S\}$ be the set of edges with both endpoints in S. For any subset $F \subseteq E$, we denote $F_R = F \cap E_R$, $F_{NR} = F \cap E_{NR}$, although we will use $\delta_R(S)$ ($\delta_{NR}(S)$) instead of $\delta(S)_R$ ($\delta(S)_{NR}$) for the required (nonrequired) edges in set $\delta(S)$. Given a vector x^d indexed on the set E of edges, let $x^d(F) = \sum_{e \in F} x_e^d$ and $(x^d + y^d)(F) = \sum_{e \in F}(x_e^d + y_e^d)$. The MDGRP can be formulated as follows:

$$\text{Minimize} \qquad \sum_{d \in \mathcal{D}} \left(\sum_{e \in E_{NR}} c_e \left(x_e^d + y_e^d\right) + \sum_{e \in E_R} c_e^s x_e^d + \sum_{i \in V_R} c_i z_i^d \right), \tag{1}$$

$$x^d(\delta_R(i)) + (x^d + y^d)(\delta_{NR}(i)) \equiv 0 \pmod 2, \ \forall i \in V \setminus V_R, \ \forall d \in \mathcal{D}, \tag{2}$$

$$(x^d + y^d)(\delta(i)) = 2\, z_i^d, \ \forall i \in V_R, \ \forall d \in \mathcal{D}, \tag{3}$$

$$x^d(\delta_R(S)) + (x^d + y^d)(\delta_{NR}(S)) \geq 2x_f^d, \ \forall d \in \mathcal{D}, \ \forall S \subseteq V \setminus \{d\}, \ \forall f \in E(S), \tag{4}$$

$$\sum_{d \in \mathcal{D}} x_e^d = 1, \ \forall e \in E_R, \tag{5}$$

$$\sum_{d \in \mathcal{D}} z_i^d = 1, \ \forall i \in V_R, \tag{6}$$

$$\sum_{e \in E_R} c_e^s x_e^d + \sum_{e \in E_{NR}} c_e \left(x_e^d + y_e^d\right) + \sum_{i \in V_R} c_i z_i^d \leq L, \ \forall d \in \mathcal{D}, \tag{7}$$

$$\sum_{i \in V_R} d_i z_i^d \leq Q, \ \forall d \in \mathcal{D}, \tag{8}$$

$$x_e^d \geq y_e^d, \ \forall e \in E_{NR}, \ \forall d \in \mathcal{D}, \tag{9}$$

$$z_i^d \in \{0, 1\}, \ \forall i \in V_R, \ \forall d \in \mathcal{D}, \tag{10}$$

$$x_e^d \in \{0, 1\}, \ \forall e \in E_R, \ \forall d \in \mathcal{D}, \tag{11}$$

$$x_e^d, y_e^d \in \{0, 1\}, \ \forall e \in E_{NR}, \ \forall d \in \mathcal{D}. \tag{12}$$

The objective function (1) minimizes the total cost of the D routes. The first term represents the "deadheading cost." The second and third terms represent the cost of servicing the required edges and the cost of visiting the required nodes, respectively, and due to constraints (5) and (6) both are constant and could be removed from (1). Constraints (2) and (3) force all vertices in each route to have even degree. The first ones imply that the edges incident with a vertex $i \in V \setminus V_R$, that is, those in $\delta_R(i) \cup \delta_{NR}(i)$, are traversed an even number of times (maybe zero) by route d, while equalities (3) ensure that route d traverses exactly twice all the (nonrequired) edges incident with a vertex i if it is serviced by this route ($z_i^d = 1$) or zero times if it is not serviced ($z_i^d = 0$).

Inequalities (4) avoid subtours and ensure the connectivity of each drone route with its depot. If a route d traverses an edge f in $E(S)$ ($x_f^d = 1$), these inequalities force the route to traverse the cut-set $\delta(S)$ at least two times in order to connect it to depot d. Otherwise ($x_f^d = 0$), these inequalities are trivially satisfied. Equations (5) force each required edge to be serviced exactly once, and the visit of all the required vertices in exactly one route is ensured by constraints (6). Constraints (7) guarantee that the cost of each route does not exceed L, while constraints (8) ensure that the total demand

serviced on each route is not greater than the drone capacity Q. Constraints (9) force that a second traversal of a nonrequired edge in a route can only occur when it has been traversed previously in the route. Constraints (10)–(12) are the binary conditions for the variables.

2.1. Valid inequalities

In what follows, we will present some families of valid inequalities that will contribute to strengthen the linear programming (LP) relaxation of the formulation (2)–(12) for the MDGRP.

2.1.1. Connectivity inequalities for required vertices

Given a depot $d \in \mathcal{D}$ and a vertex subset $S \subseteq V \setminus \{d\}$ such that $V_R \cap S \neq \emptyset$, the following connectivity inequalities are also valid:

$$x^d(\delta_R(S)) + (x^d + y^d)(\delta_{NR}(S)) \geq 2z_i^d, \quad \forall i \in V_R \cap S. \tag{13}$$

2.1.2. Parity inequalities

Constraints (2) are not linear, but they can be replaced by the following *parity inequalities*, which have been adapted from those presented in Corberán et al. (2013) for the maximum benefit Chinese postman problem, that were based on the cocircuit inequalities introduced in Barahona and Grotschel (1986):

$$x^d(\delta_R(S)\backslash F) + (x^d - y^d)(\delta_{NR}(S)\backslash F) \geq x^d(F_R) + (x^d - y^d)(F_{NR}) - |F| + 1, \tag{14}$$

for each depot $d \in \mathcal{D}$, for all $S \subset V$, and for all $F \subseteq \delta(S)$ with $|F|$ odd.

These inequalities are based on the fact that any route traverses any cut-set $\delta(S)$, with $S \subset V$, an even (or zero) number of times. Note that if a nonrequired edge e is traversed twice or not traversed in the route of depot d, $x_e^d - y_e^d = 0$ holds, and therefore the right-hand side of (14) is positive if and only if all the edges in F are traversed exactly once. Since $|F|$ is odd, at least one more edge in $\delta(S) \setminus F$ has to be traversed exactly once and then $x^d(\delta_R(S)\backslash F) + (x^d - y^d)(\delta_{NR}(S)\backslash F) \geq 1$ holds.

2.1.3. p-Connectivity inequalities

For each depot $d \in \mathcal{D}$, let $\{S_0, \ldots, S_p\}$ be a partition of V and assume that $d \in S_r$, $r \in \{0, \ldots, p\}$. For each $j \in \{0, \ldots, p\}\backslash\{r\}$, we select either an edge $e_j \in E(S_j)$ or a vertex $v_j \in S_j \cap V_R$. Let I_1 and I_2 be the sets of indices in $\{0, \ldots, p\}\backslash\{r\}$ in which an edge or a required vertex has been selected, respectively. Then, the following inequality is valid (a proof can be found in Campbell et al. (2023)):

$$x^d(\delta_R(S_0)) + (x^d + y^d)(\delta_{NR}(S_0)) + 2\sum_{1\leq q<t\leq p} x^d(S_q : S_t) \geq 2\sum_{j\in I_1} x_{e_j}^d + 2\sum_{j\in I_2} z_{v_j}^d. \tag{15}$$

These p-connectivity inequalities are a generalization to the problem with several vehicles (and depots) of those presented in Corberán et al. (2013).

2.1.4. Capacity inequalities

Given a subset of vertices $S \subset V$, let K be a lower bound on the number of drones needed to service the vertices in $S \cap V_R$. Let us denote $\mathcal{D}' = \mathcal{D} \cap S$. If $K > |\mathcal{D}'|$ then the vertices in $S \cap V_R$ cannot be serviced from the depots in $\mathcal{D}'$ and at least $K - |\mathcal{D}'|$ drones from depots outside of S must enter in S, thus crossing the cut-set $\delta(S)$. Therefore, the following *capacity inequality* is valid for the MDGRP:

$$\sum_{d \in \mathcal{D} \setminus \mathcal{D}'} x^d(\delta_R(S)) + \sum_{d \in \mathcal{D} \setminus \mathcal{D}'} (x^d + y^d)(\delta_{NR}(S)) \geq 2\big(K - |\mathcal{D}'|\big), \tag{16}$$

An easy bound K on the number of drones needed to service the required vertices in a set S is

$$K = \Big\lceil \sum_{i \in V_R \cap S} d_i / Q \Big\rceil.$$

If, for example, $K - |D'| = 2$, then at least two *different* drones must cross the cut-set $\delta(S)$ and the following *capacity inequalities* are also valid for the MDGRP:

$$\sum_{d \in \mathcal{D} \setminus (\mathcal{D}' \cup \{b\})} x^d\,(\,\delta_R\,(\,S)) + \sum_{d \in \mathcal{D} \setminus (\mathcal{D}' \cup \{b\})} (\,x^d + y^d)\,(\,\delta_{NR}\,(\,S)) \geq 2\,\big(K - |\mathcal{D}'| - 1\big)\,\forall b \in \mathcal{D} \setminus \mathcal{D}'.$$

3. A branch-and-cut algorithm for the multidepot general routing problem

In this section, we describe the branch-and-cut algorithm we have implemented for the MDGRP based on the formulation presented in Section 2. The initial LP is defined by equalities (3), (5), and (6), inequalities (7)–(9), and the bounds on the variables. Connectivity inequalities (4) and (13) are exponential in number and are separated at each iteration of the cutting-plane algorithm and added to the LP. Moreover, the previous parity, p-connectivity, and capacity inequalities are separated with the algorithms described in the below section.

3.1. Separation algorithms

At each iteration of the cutting-plane procedure, we use some *separation algorithms* to identify valid inequalities that may be violated by the current LP solution. Let $(\bar{x}, \bar{y}, \bar{z})$ denote this solution, possibly fractional. In the following, we describe the separation procedures used for each family of valid inequalities, in which ε will correspond to a given parameter that is used in some of these procedures.

3.1.1. Connectivity inequalities (4) and (13)

Connectivity inequalities (4) and (13) for each depot d can be exactly separated in polynomial time. Given the support graph induced by the edges $e \in E$ such that $\bar{x}_e^d > 0$, we calculate a max-flow from d to each edge f or required vertex i such that $\bar{x}_f^d > 0$ or $\bar{z}_i^d > 0$. We use this algorithm

only for identifying violated inequalities of type (13) because the number of required vertices in our instances is not large. However, in what refers to inequalities (4), the exact procedure is time consuming and usually produces many violated inequalities that are very similar to each other. Therefore, we have decided in this case to use heuristic algorithms for separating them.

The first heuristic is well known and consists of the computation of the connected components of the graph induced by the edges $e \in E$ such that $\bar{x}_e^d + \bar{y}_e^d > \varepsilon$, plus the depot d, if necessary. The inequalities (4) and (13) associated with each connected component and with the edge f with the maximum $\bar{x}_f^k$ or the required vertex i with the maximum $\bar{z}_i^d$ are checked for violation.

The second heuristic works in a smaller graph in which the connected components of the graph induced by edges with $\bar{x}_e^d \geq 1 - \varepsilon$ are shrunk into a single vertex each. Then, all minimum cuts between the vertex corresponding to the component containing the depot d and the remaining ones are calculated, and their corresponding connectivity inequalities (4) and (13) are checked for violation.

3.1.2. Parity inequalities (14)

Given a subset $S \subset V$, Ghiani and Laporte (2000) described a procedure for choosing the subset of edges $F \subseteq \delta(S)$ that produces the most violated co-circuit inequality, if any, associated with this cut-set. Given depot $d \in \mathcal{D}$, since parity inequalities (14) are derived from co-circuit inequalities, we can adapt this method as follows.

Each edge $e \in \delta_{NR}(S)$ with $\bar{x}_e^d - \bar{y}_e^d > 0.5$ and each $e \in \delta_R(S)$ with $\bar{x}_e^d > 0.5$ are included in F. If $|F|$ is odd, we already have the optimal set F, and we proceed to check the corresponding parity inequality for violation. If $|F|$ is even, we need to add one more edge to F or remove one from it. We look for the edge in F with the lowest value of $\bar{x}_e^d - \bar{y}_e^d$ (or $\bar{x}_e^d$) and the edge in $\delta(S) \setminus F$ with the highest value of $\bar{x}_e^d - \bar{y}_e^d$ (or $\bar{x}_e^d$), and choose the best option between removing the first one from F and adding the second one to it. Then we check the parity constraint for violation.

We apply this procedure for all the depots $d \in \mathcal{D}$ and all the subsets S formed by one single required vertex as well as for general subsets S obtained by computing the connected components of the graph induced by the edges $e \in E_{NR}$ with $\bar{x}_e^d - \bar{y}_e^d > \varepsilon$ and those in $e \in E_R$ with $\bar{x}_e^d > \varepsilon$, plus the depot d, if necessary.

3.1.3. p-Connectivity inequalities (15)

To separate these inequalities, we consider the connectivity inequalities (4) found by the connectivity separation procedures that are satisfied with equality. Let S be the set of vertices associated with one of these tight inequalities and assume that $d \in S_0 = V \setminus S$. Now we compute the connected components of the subgraph induced in $G(S)$ by those edges $e \in E(S)$ such that $\bar{x}^d(e) \geq 1 - \varepsilon$. For each connected component C_j, we define $w_j = \max\{\bar{x}_e^d : e \in E(V_j),\ \bar{z}_i^d : i \in V_j \cap V_R\}$, and for each pair C_i, C_j, $s_{ij} = 2\bar{x}^d(V_i : V_j) - 2\min\{w_i, w_j\}$, where V_t denotes the set of vertices in C_t. The components that maximize s_{ij} are iteratively shrunk while s_{ij} is positive. In this way, we obtain sets $S_1, \ldots, S_p$ and the corresponding inequality (15) is checked for violation.

3.1.4. Capacity inequalities (16)

Capacity inequalities (16) can be separated heuristically by using the following simple procedure. First, we build the support graph $\overline{G} = (V, E_R \cup \overline{E}_{NR})$, where $\overline{E}_{NR}$ is defined by those nonrequired edges with $\bar{c}_e = \sum_{d \in \mathcal{D}} (\bar{x}_e^d + \bar{y}_e^d) > 0$. Required edges have unit capacity, and edges in $\overline{E}_{NR}$ have capacity $\bar{c}_e$. For each depot $d \in \mathcal{D}$ and each required vertex $i \in V_R$, we compute the maximum flow in $\overline{G}$ from d to i. Let $(S : V \setminus S)$ be the associated minimum cut, with $i \in S$. We define $\mathcal{D}' = \mathcal{D} \cap S$ and $K = \left\lceil \sum_{j \in V_R \cap S} d_j / Q \right\rceil$. If the maximum flow is less than $2\,(K - |\mathcal{D}'|)$, this cut-set defines a violated capacity inequality (16).

3.2. The cutting-plane algorithm

At each iteration, the cutting-plane algorithm applies, for each depot $d \in \mathcal{D}$, the first heuristic algorithm for connectivity inequalities with $\varepsilon = 0, 0.25, 0.5$, where the value of ε is increased only if no violated inequalities are found with the previous value. Moreover, the second connectivity heuristic is applied with $\varepsilon = 0, 0.1, 0.2, 0.3, 0.4$. All the tight cut-sets obtained with the connectivity separation procedures are stored and used by the p-connectivity heuristic with $\varepsilon = 0, 0.15, 0.3$. If the above connectivity separation heuristics fail in finding violated cuts for the route associated with a given depot d, the algorithm based on the exact separation procedure for connectivity inequalities (13) is applied for this depot.

Moreover, at each iteration, we apply the exact parity separation procedure for single vertices and the heuristic procedure for parity inequalities with $\varepsilon = 0, 0.25, 0.5$. The heuristic for identifying violated capacity inequalities is applied only for depots for which no violated connectivity inequalities have been found and only at nodes of the search tree whose depth is less than or equal to four.

This cutting-plane algorithm is applied at each node of the search tree whose depth is less than 9 while new violated inequalities are found and the value of the objective function has increased at least $\varepsilon_{off}\%$ in the last 20 iterations, where $\varepsilon_{off} = 0.02$ in the root node and $\varepsilon_{off} = 0.05$ in the remaining ones. Since we have observed that the number of cuts found decreases with the depth of the node in the tree, we have decided not to use the cutting-plane algorithm for nodes of depth 9 or greater and just branch.

4. Matheuristic algorithms for the MDdGRP

In this section, we present two matheuristic algorithms for solving the MDdGRP. Each MDdGRP instance contains a set of required lines, a set of required locations, and a set of depots. Each line of an MDdGRP instance is given by a polygonal chain with a large number of points and segments that can be considered a very good approximation to the original line. If we assume that drones can only enter and exit each line through the points of the polygonal chain, we have an instance of the MDGRP, a discrete optimization problem. As the polygonal chains contain a large number of points, the solutions of this MDGRP instance can be considered a very good approximation to the solutions of the original MDdGRP instance. However, this large number of points makes the

MDGRP instance usually too difficult to solve. The main idea of the procedures we propose here is to work with smaller MDGRP instances by using only a few *intermediate* points of each line that we consider the most promising ones. Thus, each MDGRP instance contains, for each line, its two endpoints and a few intermediate points, perhaps none. The segments joining these points define the required edges of the MDGRP instance, with a cost for each segment proportional to its length, so that the sum of these costs is the service cost of the original line. Note that the addition or removal of intermediate points in a given MDGRP instance generates a different MDGRP instance.

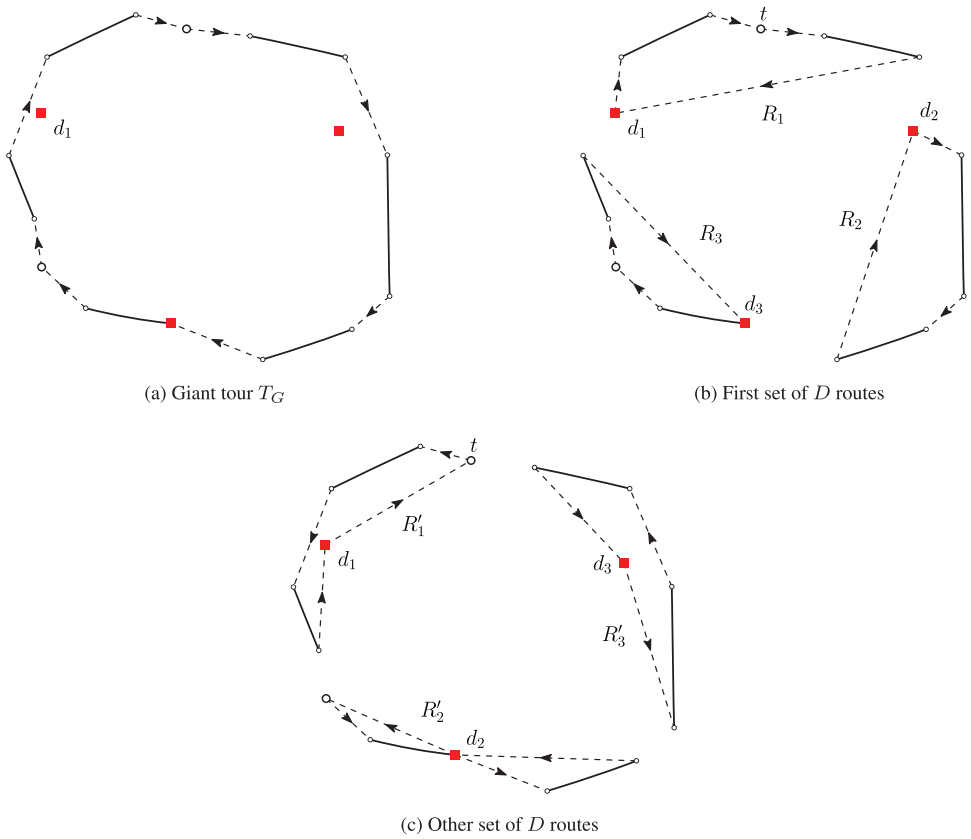
The two algorithms we present combine a construction phase based on an order–first split–second procedure, which is described in Section 4.1, four local search procedures embedded in a variable neighborhood descent (VND) algorithm for improving the solutions, described in Section 4.2, and a set of rules to choose the intermediate points that will be added to the MDGRP instance, detailed in Section 4.3. The overall algorithms proposed are described in Section 4.4.

4.1. Construction phase

Let us consider an MDGRP instance defined on an undirected graph $G = (V, E)$. The first step in order to find a set of initial MDGRP solutions is to compute a giant tour T_G on G traversing all the required edges and visiting all the required vertices. This is done by defining a GRP instance on G with the same sets of required edges and required vertices as the MDGRP instance and solving it optimally with the branch-and-cut algorithm proposed by Corberán et al. (2007). This solution of the GRP instance provided by this exact algorithm is an Eulerian graph, from which a giant (directed) tour T_G is obtained by applying the algorithm proposed by Hierholzer (1873). The giant tour T_G determines an ordered sequence of the tasks (required vertices and edges) but does not satisfy the limitations of capacity and duration of the routes. Therefore, we consider several ways of splitting it into D routes, each of them starting and ending at a different depot of $\mathcal{D}$, by following the procedure described below. Figure 2a illustrates a (directed) tour on an instance with three depots (represented by red squares), two required vertices, and six required edges. Note that one depot is incident with a required edge while the other two are isolated.

Let $\{t_1, t_2, \ldots, t_{|E_R|+|V_R|}\}$ be the sequence of tasks (required vertices and edges) in the order determined by the giant tour T_G. First, we select a depot $d_1 \in \mathcal{D}$ (Fig. 2a). The first route R_1 starts at d_1 and deadheads to the closest vertex. From there, R_1 follows the giant tour T_G while the duration (taking into account the time of the way back to the depot) and capacity limits are satisfied. Let t_i be the last task of T_G performed by R_1. Then, a new route R_2 starts at the depot of $\mathcal{D} \setminus \{d_1\}$ closest to task t_{i+1}. It continues following T_G until adding one more task would make R_2 infeasible. This procedure is repeated until $D - 1$ routes have been generated. The last route R_D, which is rooted at the last nonused depot, includes all the tasks not serviced in any previous route in the same order as T_G (see Fig. 2b). Note that this solution (set of D routes) may not be feasible due to the last route violating the capacity or duration constraints. If this happens, we apply the local search procedures described in Section 4.2 to this solution in order to try to make it feasible. If, after applying these methods, the solution is still not feasible, we discard it.

Now consider depot d_1, and the set of tasks performed in route R_1. We select a task t in R_1 and generate a new solution by starting the first route at d_1, going directly to task t, following the giant



(a) Giant tour T_G

(b) First set of D routes

(c) Other set of D routes

Fig. 2. Construction of D routes from T_G.

tour T_G in the opposite direction, and splitting T_G into routes as described above (see Fig. 2c). By choosing a different task t of R_1 each time, we generate several different solutions.

We repeat the construction phase explained above by selecting each depot in $\mathcal{D}$ as starting depot d_1. Let us call $\bar{S}$ to the set of feasible MDGRP solutions generated by this construction phase.

4.2. Local search procedures

To improve a given MDGRP solution, or trying to transform an infeasible solution into an MDGRP solution, we have implemented the following local search procedures that iteratively explore different ways of exchanging tasks within a route and between routes:

(*i*) *Intraroute move*. This first procedure is applied to each route R in order to try to reduce its cost. Iteratively, each task in R is moved to the position of that route that minimizes its total cost until no improvements can be found.

(*ii*) *Destroy and repair move*. This second method randomly selects n tasks among all the routes, where n is a random number between 2 and 8, and removes them from their routes. Then, for

each removed task, we evaluate the cost of inserting this task in all the possible routes and positions and insert it in the route and position that minimizes this cost. If the new solution obtained is not better than the starting one, or if it is not possible to insert a task in any route without exceeding Q or L, the changes are discarded and a new iteration is executed. This procedure stops when it_{max} consecutive iterations that do not improve the current solution are performed, where it_{max} is a given parameter.

Given a parameter ℓ_{max}, the following two methods use a *first improvement* strategy by swapping chains of at most ℓ_{max} consecutive tasks between two different routes.

(*iii*) *0 to ℓ exchange*. Starting with $\ell = 1$, this procedure explores the removal of all the possible chains of ℓ consecutive tasks from a route and its insertion into another route and position that minimizes the total cost (and does not lead to infeasible routes). ℓ is incremented by 1 if no improved solution is found and reset to 1 each time the current solution is improved. If ℓ reaches ℓ_{max} and no move improves the current solution, the procedure stops.

(*iv*) *ℓ_1 to ℓ_2 exchange*. Similarly to the previous one, this procedure explores the exchange of a chain of ℓ_1 consecutive tasks of a route with another of l_2 consecutive tasks from another route. We start with $\ell_1 = \ell_2 = 1$ and then increase ℓ_2 by one unit at a time. When ℓ_2 reaches ℓ_{max} (without improvement), we increase ℓ_1 by one and reset ℓ_2 to 1. We proceed in this way until an improvement is achieved (and then both parameters are reset to 1) or $\ell_1 = \ell_2 = \ell_{max}$.

Given the set of solutions obtained in the construction phase of Section 4.1, we apply these four local search procedures to those solutions that did not satisfy the capacity or duration constraints to try to transform them into feasible solutions. When evaluating each possible move of the local search procedures, we first check if it turns an infeasible route to a feasible one. If this happens, we apply this movement even if the total cost worsens.

In order to improve the MDGRP solutions in $\bar{S}$, we have embedded the above improvement procedures into a VND algorithm (see Hansen and Mladenović, 2001).

Once the VND procedure has been performed, an optimization procedure is applied to each route of the solutions in $\bar{S}$. For each route R, we define a GRP instance on the graph induced in G by the required edges and vertices serviced by R plus its depot, if necessary. This GRP instance is solved to optimality with the branch-and-cut algorithm proposed in Corberán et al. (2007).

4.3. Selection of intermediate vertices

Let us denote by $\mathcal{C}$ the set of polygonal chains of the "original" MDGRP instance and let $V_{\mathcal{C}} = V_E \cup V_P$ be the set of vertices incident with the segments of $\mathcal{C}$, where V_E are those vertices that are endpoints of the original lines and V_P are the intermediate points. Given a subset $U \subseteq V_{\mathcal{C}}$, we define $V_{\mathcal{C}}(U)$ as the union of the vertices of all the polygonal chains that have some vertex in U. In order to create a smaller MDGRP instance, we will select only a subset of intermediate vertices, which will be called $\mathcal{I}$, after applying the following rules:

R1 For each depot $d \in \mathcal{D}$, we first select the vertex $v_1 \in V_{\mathcal{C}} \setminus V_{\mathcal{C}}(\{d\})$ closest to d and the vertex $v_2 \in V_{\mathcal{C}} \setminus V_{\mathcal{C}}(\{d, v_1\})$ closest to d, and they are added to $\mathcal{I}$ if they belong to V_P. If v_1 or v_2

belong to V_E, we try with a third vertex $v_3 \in V_C \setminus V_C(\{d, v_1, v_2\})$ closest to d and is added to $\mathcal{I}$ if it belongs to V_P.

R2 For each required vertex $i \in V_R$, we proceed as in R1, with $v_1 \in V_C$, $v_2 \in V_C \setminus V_C(\{v_1\})$ and $v_3 \in V_C \setminus V_C(\{v_1, v_2\})$.

R3 For each vertex $v \in V_E$, we study candidate points v_1, v_2, and v_3 as in R1, but now only the first vertex belonging to V_P is added to $\mathcal{I}$.

The previous rules generate an MDGRP instance, which we will denote MDGRP($\mathcal{I}$), in which the set of vertices is $V = V_R \cup V_E \cup \mathcal{I}$. In subsequent steps of the matheuristic algorithms, once a set of solutions has been obtained, further intermediate vertices can be added as follows:

R4 Given a set of solutions of MDGRP($\mathcal{I}$), for each $u \in \mathcal{I}$ incident with a nonrequired edge used in any of the solutions, we proceed as in R3, thus generating a new set $\mathcal{I}' \supseteq \mathcal{I}$.

4.4. Overall algorithms

We describe here the algorithms we have developed for solving the MDdGRP that combine all the procedures detailed in Sections 4.1–4.3.

4.4.1. Algorithm Mat1

The first matheuristic (Mat1) applies first the rules R1–R3 described in Section 4.3 for generating the MDGRP($\mathcal{I}$) instance, containing already some intermediate points, and then performs the construction phase of Section 4.1 in order to obtain a set of initial solutions. These solutions are improved with the VND and the route optimization methods described in Section 4.2. The resulting solutions form a set $\bar{S}_0$.

Let $\bar{S}_1$ be the set of the 10 best solutions in $\bar{S}_0$. With this set, the algorithm computes now a new MDGRP($\mathcal{I}'$) instance by following rule R4 described in Section 4.3. Each solution $s' \in \bar{S}_1$ is transformed into an MDGRP($\mathcal{I}'$) solution s'' as follows. Each required edge (i, j) of s' is replaced by $p_{ij} + 1$ required edges $(i, i_1), (i_1, i_2), \ldots, (i_{p_{ij}}, j)$, where p_{ij} denotes the number of intermediate points of the original line joining i and j that are included in MDGRP($\mathcal{I}'$). The VND and route optimization procedures of Section 4.2 are applied again to the adapted solutions, and the best MDGRP($\mathcal{I}'$) solution obtained is selected as the output of Mat1.

4.4.2. Algorithm Mat2

The second matheuristic (Mat2) starts by performing the construction phase of Section 4.3 to the MDGRP(0) instance (the MDGRP instance in which each original line is represented by a single required edge with no intermediate points) to obtain an initial set of solutions. Then each MDGRP(0) solution is improved with the VND algorithm and the route optimization procedure. The resulting MDGRP(0) solutions form a set $\bar{S}_0$.

Next, the MDGRP($\mathcal{I}$) instance is generated by following the rules R1–R3. This instance incorporates some new intermediate points. The best 10 solutions in $\bar{S}_0$ are transformed into an

MDGRP($\mathcal{I}$) solution as explained in MAT1. All the transformed solutions are improved with the VND and the route optimization procedures and define a new set $\bar{S}_1$.

Rule R4 is applied using the set $\bar{S}_1$ to obtain a new MDGRP($\mathcal{I}'$) instance. Each solution in $\bar{S}_1$ is transformed into an MDGRP($\mathcal{I}'$) solution as explained above, and the VND and route optimization procedures are applied again. The best MDGRP($\mathcal{I}'$) solution obtained is selected as the final solution of MAT2.

Note that, although the same name is used, the MDGRP($\mathcal{I}'$) instances generated by algorithms MAT1 and MAT2 are not necessarily the same.

5. Computational experiments

In this section, we describe the computational experiments we have run to analyze the performance of the proposed branch-and-cut and matheuristic algorithms.

Since there were no available MDdGRP benchmark instances, we have created a set of instances with a set of depots, a set of required vertices, and a set of required lines. The algorithms have been coded in C++, and all the tests have been run on an Intel Core i7 at 3.4 GHz with 32 GB RAM.

For the implementation of the branch-and-cut algorithm, we have used IBM Ilog Cplex 12.10 with Concert Technology. Cplex heuristic algorithms were turned off, and Cplex's own cuts were activated in automatic mode. The optimality gap tolerance was set to zero, the best bound strategy was used for the tree exploration, and the strong branching strategy was used for choosing the variable for branching.

5.1. Instances

In order to test the solution methods proposed, we have randomly generated some multidepot drone GRP instances (networks with families of polygonal chains, required vertices, and several depots) on a grid as follows.

First, we generate a GRP instance on a grid with $m \times n$ points whose coordinates are multiples of 100. The graph initially contains all the vertical and horizontal edges of the grid as well as some random diagonals. We randomly select the location of *ndep* depots among the nodes of the grid. Each edge is considered required with probability p. The vertices, except for the depots, that are not incident with required edges are now removed. Following the procedure described in Campbell et al. (2018) for the drone rural postman problem, the coordinates of all the vertices are slightly perturbed, the shape of the required lines is altered according to parameter *curvature*, and a series of vertices are added splitting each required line into segments of similar length. We now generate a set of *nvreq* required vertices with random coordinates. If the Euclidean distance between a required vertex and the closest vertex is less than a certain value, the required vertex is removed and a new one is generated. Each required vertex i is assigned a demand d_i chosen at random in $\{1, 2\}$. The deadheading cost c_{ij} between any two vertices i, j is equal to the Euclidean distance between them. The cost of each segment of a line is computed as the Euclidean distance between its endpoints multiplied by 1.5, and the cost of the line is equal to the sum of the costs of its segments.

A total of 15 different grid sizes have been considered, ranging from 5×6 to 10×10. Parameter *ndep* has been chosen in {2, 3, 4} for grids with less than 50 nodes, {3, 4, 5} for those with between 50 and 70 nodes, and {4, 5, 6} for grids with more than 70 nodes. For each grid size and value of *ndep*, two instances have been generated. The first one is with $p = 0.3$, $curvature = 0.4$, and *nvreq* a random number in $[3\times ndep, 4\times ndep]$ (version 1), and the second one is with $p = 0.4$, $curvature = 0.3$ and *nvreq* in $[2\times ndep, 3\times ndep]$ (version 2). We have a total of 90 instances, which we have grouped into three groups: small, with less than 50 nodes, medium with between 50 and 70 nodes, and large, with more than 70 nodes.

For each instance, the maximum duration L and the load limit Q of the drones are calculated as follows:

- $L = \frac{\text{cost}(T_G)}{D} + \max\{c_{ij} : i, j \in V\}$ rounded up to a multiple of 100.
- $Q = \left\lceil \frac{\sum_{i\in V_R} d_i}{D} \right\rceil + 2$.

If an instance is infeasible with these values of L and Q, they are slightly increased until it becomes feasible.

Table 1 summarizes the characteristics of the MDdGRP instances. Each row corresponds to a set of three different instances, each one with a different number of depots. The digits at the end of the name on the first column indicate the values of parameters m and n and whether the instances have been generated as version 1 or 2. Thus, for example, the instances grouped as "MDdGRP5_10_2" were generated with $m = 5$, $n = 10$, and version 2 ($p = 0.4$, $curvature = 0.3$, and *nvreq* in $[2\times ndep, 3\times ndep]$). Column "Depots" shows the number of depots used for the instances of each group. Columns 3 and 4 report the average number of required vertices and lines, respectively, of the three instances, while column 5 shows the average number of vertices not considering the ones generated to split the required lines. These 90 instances, as well as the results obtained in the experiments explained in what follows, can be found at http://www.uv.es/plani/instancias.htm.

5.2. Computational results

We have used a representative subset of 10 MDdGRP instances with different sizes and number of depots in order to choose the values for the parameter it_{max} (the maximum number of consecutive iterations without improvement in the destroy and repair procedure) and ℓ_{max} (the maximum number of tasks that can be considered in an exchange move). We have tried all the combinations of values for it_{max} in {10, 20, 30, 40} and for ℓ_{max} in {5, 10, 20, 30, $\mathcal{L}$}, where the value $\mathcal{L}$ is computed for each instance as $\mathcal{L} = \lceil(|E_R| + |V_R|)/D\rceil$. The computational results obtained with each combination of these values on the subset of instances are reported in Table 2. In this table, column 3 shows the average computing time, in seconds, used to solve the instances with the parameters indicated in columns 1 and 2, while column 4 shows the total cost on average. The "Gap" column provides the average percentage gap between the solution obtained with the parameters of its corresponding row and the best solution found among all the considered combinations. Since the average computing time is not much larger and they provide the best deadheading cost and gap, we have chosen $it_{max} = 30$ and $\ell_{max} = 20$ as the values that will be used in the matheuristic algorithms in the following computational experiments.

Table 1
Characteristics of the multidepot drone GRP instances

Instance name	Depots	Required vertices	Required lines	Vertices
MDdGRP5_6_1		9.0	17.0	32.0
MDdGRP5_6_2		10.3	22.3	36.7
MDdGRP6_6_1		9.0	21.0	36.0
MDdGRP6_6_2		10.0	29.7	43.0
MDdGRP5_8_1	2, 3, 4	9.0	25.0	42.0
MDdGRP5_8_2		9.7	36.3	44.7
MDdGRP6_8_1		9.0	29.0	47.0
MDdGRP6_8_2		10.0	48.3	54.0
MDdGRP7_7_1		9.0	30.0	48.0
MDdGRP7_7_2		11.0	47.7	54.3
MDdGRP5_10_1		12.0	30.0	50.0
MDdGRP5_10_2		13.7	49.7	60.3
MDdGRP6_10_1		12.0	40.0	65.0
MDdGRP6_10_2		13.3	56.0	67.3
MDdGRP7_9_1	3, 4, 5	12.0	45.0	68.7
MDdGRP7_9_2		13.3	58.7	67.3
MDdGRP8_8_1		12.0	45.0	64.0
MDdGRP8_8_2		13.7	60.7	72.0
MDdGRP7_10_1		12.0	51.0	71.3
MDdGRP7_10_2		13.0	64.0	71.0
MDdGRP8_9_1		15.0	53.0	77.0
MDdGRP8_9_2		15.7	63.3	78.0
MDdGRP9_9_1		15.0	55.0	79.0
MDdGRP9_9_2		17.0	69.3	85.3
MDdGRP8_10_1	4, 5, 6	15.0	60.0	89.0
MDdGRP8_10_2		16.0	73.0	92.0
MDdGRP9_10_1		15.0	55.0	82.3
MDdGRP9_10_2		17.7	69.7	87.0
MDdGRP10_10_1		15.0	67.0	100.7
MDdGRP10_10_2		15.7	86.3	99.0

Table 3 reports the computational results obtained with algorithms MAT1 and MAT2 described in Section 4.4. The instances have been grouped into three blocks labeled "Small," "Medium," and "Large" according to their size, with 30 instances in each group. Each row presents the average results obtained for the 10 instances with the same number of depots, which is given in the second column. Columns 4–7 contain the results obtained by MAT1, and columns 8–11 are those obtained by MAT2. Both blocks show the average total cost for the 10 instances with the same characteristics, the average percentage gap between the best cost known (considering both algorithms) and the cost obtained with the corresponding algorithm, the number of times (♯ best) that the algorithm has found the best solution, and the average computing time (in seconds). The last row provides the average of the columns except for the number of best solutions found, that is the sum of them.

It can be seen that algorithm MAT2 outperforms MAT1, obtaining the best solution in 64 of the 90 instances, while the average computing time is very similar for both methods. The difference

Table 2
Sensitivity of the matheuristics to the parameters it_{max} and ℓ_{max}

$\mathbf{it_{max}}$	$\boldsymbol{\ell_{max}}$	Time	Cost	Gap
	5	25.85	13,781.32	1.54
	10	25.59	13,663.46	0.84
10	20	32.91	13,583.08	0.34
	30	31.43	13,596.81	0.45
	$\mathcal{L}$	30.66	13,605.37	0.50
	5	17.82	13,785.75	1.61
	10	23.48	13,687.60	0.99
20	20	29.67	13,578.20	0.33
	30	29.85	13,583.80	0.36
	$\mathcal{L}$	26.84	13,652.38	0.74
	5	16.58	13,816.64	1.73
	10	22.98	13,668.09	0.85
30	**20**	**30.02**	**13,577.11**	**0.31**
	30	31.06	13,595.96	0.42
	$\mathcal{L}$	27.83	13,611.80	0.51
	5	18.37	13,824.73	1.93
	10	24.10	13,709.49	1.13
40	20	31.68	13,617.25	0.54
	30	30.61	13,581.67	0.33
	$\mathcal{L}$	27.52	13,616.88	0.57

Table 3
Results obtained with algorithms MAT1 and MAT2

			MAT1				MAT2			
Size	D	♯ inst	Cost	Gap	♯ best	Time	Cost	Gap	♯ best	Time
Small	2	10	7,925.91	0.02	8	35.13	7,930.24	0.09	8	17.58
	3	10	8,720.39	0.18	7	16.67	8,726.37	0.22	5	13.12
	4	10	9,515.99	0.68	4	13.89	9,453.31	0.10	8	15.83
Medium	3	10	12,351.82	0.03	7	61.61	12,363.74	0.11	6	41.16
	4	10	13,148.02	0.46	5	44.16	13,119.80	0.15	8	35.97
	5	10	14,015.02	0.96	4	38.54	13,936.70	0.34	6	33.62
Large	4	10	17,044.32	0.84	5	49.48	16,918.60	0.19	5	73.10
	5	10	17,391.29	0.96	1	56.59	17,230.81	0.00	9	83.59
	6	10	18,331.65	0.76	1	118.75	18,216.29	0.11	9	93.27
Total			13,160.49	0.54	42	48.31	13,099.54	0.15	64	45.25

between these two procedures is particularly evident in the larger instances and those with a larger number of vehicles.

From these results, it seems that it is a better strategy to start by applying the matheuristic to a smaller instance with few or no intermediate nodes and then incorporate new intermediate points wisely (MAT2), rather than adding many intermediate points from the start (MAT1).

Rule R4 of the matheuristic algorithms adds new intermediate points to the instance based on the intermediate points used by the previously constructed solutions. One could think about

Table 4
Effect of rule R4

	Before R4		After R4			
	Cost	Time	Cost	Time	Imp (%)	♯ imp
MAT1	13,209.73	37.03	13,160.49	48.31	0.32	46/90
MAT2	13,104.33	32.50	13,099.54	45.25	0.03	11/90

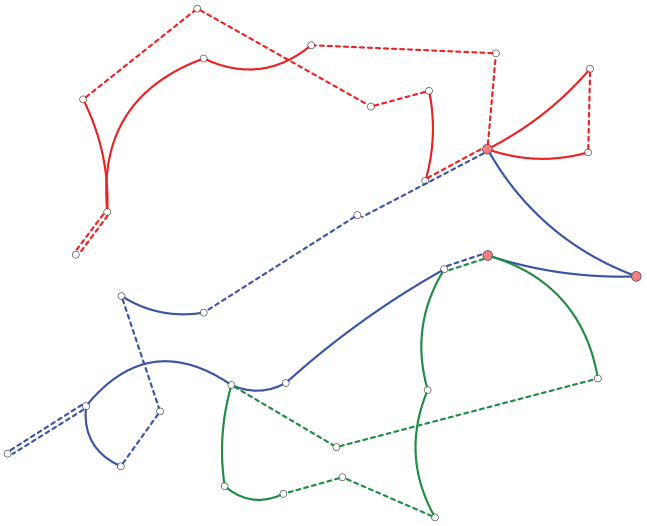
Table 5
Results with the B&C on the MDGRP(0) instances

Size	D	♯ inst	♯ opt	Gap 0	Gap	♯ imp UB	Time
Small	2	10	9	3.34	0.19	1	9.95
	3	10	7	4.52	0.51	4	147.67
	4	10	7	4.57	1.19	4	542.04
Medium	3	10	3	4.06	1.62	3	113.49
	4	10	3	4.64	2.55	3	1383.53
	5	10	0	5.50	3.94	0	–
Large	4	10	0	5.07	4.06	0	–
	5	10	0	5.54	4.55	0	–
	6	10	0	5.13	4.52	0	–

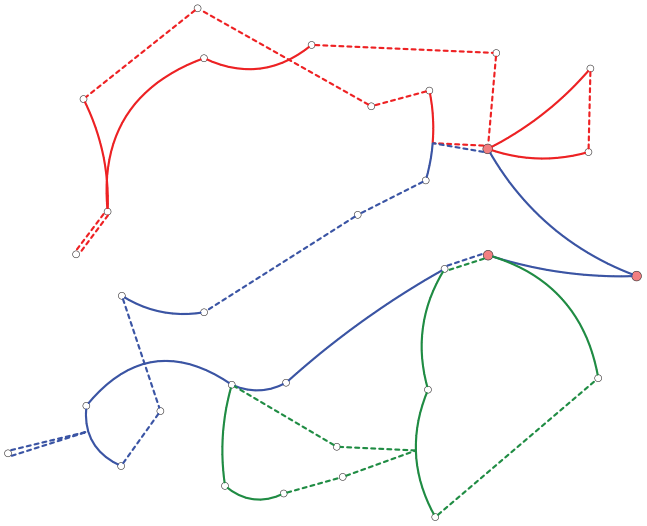
applying this rule again to the new solutions obtained in this way in order to try to find new intermediate vertices that could improve the solutions even further. To study if adding one more round of intermediate vertices would help, we have studied the effect of rule R4 on the quality of the solutions. Table 4 shows the average cost of the solutions obtained for all the 90 instances with and without applying rule R4 as well as the average computing time. The last two columns report the average improvement after applying R4, and the number of instances in which an improvement was observed. As can be seen, the improvement in the case of matheuristic MAT2 is very small, so we do not think that adding more intermediate vertices applying this rule again will be worth the computational effort. The improvement observed in MAT1 is a bit larger, but the average value of the solutions is still behind MAT2 even without R4.

Figure 3a and b illustrates an example of a solution obtained with MAT2 before and after adding the promising intermediate points. Figure 3a shows the solution without intermediate points, and the deadheading cost is 2265.83. Figure 3b depicts the solution with intermediate points obtained after applying rules R2–R4, and its deadheading cost is 2221.02. Adding intermediate points allows us to reduce the deadheading cost by 1.98%. In the new solution, there are three lines in which drones enter at an intermediate point. Moreover, the service of one of these lines is shared by two drones.

In order to assess the performance of the branch-and-cut algorithm presented in this paper, we have tested it on the instances described above without using intermediate points, that is, MDGRP(0). The exact algorithm uses as the initial upper bound the one obtained in the first phase of the heuristic algorithm MAT2 without using intermediate points. The time limit has been set to 1 hour. The results obtained are reported in Table 5. As above, the instances have been grouped by size. Column "♯ opt" reports the number of instances for which the optimal solution has been found within the time limit. "Gap 0" and "Gap" columns provide the average percentage



(a) Before adding intermediate points



(b) After adding intermediate points

Fig. 3. A solution before and after adding the promising intermediate points.

gap between the best upper bound found by the procedure and the lower bound obtained at the end of the root node and the final lower bound, respectively. The number of instances for which the branch-and-cut algorithm (B&C) has been able to improve on the initial upper bound is given in

Table 6
MAT2 performance compared with MDGRP(0) bounds

		MDGRP(0)		MDdGRP
Size	D	Gap LB (%)	♯ opt	DH Red. (%)
Small	2	0.26	8	−1.07
	3	1.13	2	0.84
	4	1.42	3	−0.20
Medium	3	1.65	3	−0.97
	4	2.64	0	−0.82
	5	3.94	0	−2.21
Large	4	4.06	0	−1.53
	5	4.55	0	−2.94
	6	4.52	0	−1.20
Total		2.69	16/90	−1.11

column labeled "♯ imp UB," while the last column shows the average computing time, in seconds, for those instances that have been solved optimally.

It can be observed that the branch-and-cut algorithm is capable of solving most of the small-size instances (23 out of 30), but it struggles to solve the medium-size ones (only 6 out of 30) and cannot solve any of the large ones. The final gap, which is in general small, also presents a similar behavior, increasing from 0.19% in the smallest instances to 4.55% in the largest ones. Regarding the gap at the root node, although a small increase can also be observed, this increase is smaller, only from 3.34% to 5.54%. This seems to indicate that the lower bounds obtained at the root node are quite good, specially considering that, for the largest instances, the branch-and-cut algorithm is not able to obtain any new upper bounds, so the comparison is made against the solution provided by the heuristic, which may still be far from the optimal one.

Table 6 compares the results obtained with the best matheuristic, MAT2, with the lower and upper bounds obtained by the branch and cut for MDGRP(0). The results are grouped according to the size of the instances and the number of depots as in previous tables. Column "Gap LB(%)" shows the average gap between the solution for the MDGRP(0) instance with respect to the lower bound obtained by the branch and cut, while column "♯ opt" reports the number of instances for which the solution of MAT2 is optimal. We can observe that the performance of the matheuristic is very good, being capable of obtaining solutions that are guaranteed to be optimal in 16 instances, while for those that are not proven to be optimal, the gaps are small.

The last column of Table 6, labeled "DH Red.(%)," presents the average percentage reduction of the deadheading cost obtained by the final solution of MAT2 compared to the deadheading cost of the best solution found by the branch and cut for the MDGRP(0) instance (negative values mean that there is an improvement). With these values, we want to assess the improvement obtained by using intermediate points in the edges versus just solving the instances without them. Note that, since these are heuristic solutions, some of them can be worse than the optimal solutions obtained for the MDGRP(0) instances. This behavior can be observed in the second row, where we get a positive average value. As can be seen, the average improvement is 1.11%, although in some sets of instances this value comes close to 3%. Therefore, we can conclude that we can use this matheuristic algorithm to obtain better solutions that take profit of the possibility of using intermediate points

while using small computation times. Even in many of the instances where the optimal solution of MDGRP(0) is known, such as the small instances with two depots, the matheuristic is capable of obtaining better solutions.

6. Conclusions

In this paper, we have addressed the MDdGRP with duration and capacity constraints, a continuous optimization problem which is an extension of the classical GRP incorporating multiple depots and a drone located in each one of them. The objective is to collectively perform all services, visiting the required nodes and traversing the network lines, while minimizing the total flight duration considering drone capacity and range.

We have presented an integer formulation for the MDGRP, a discrete version of the problem, along with some families of valid inequalities that strengthen this formulation. We have implemented a branch-and-cut procedure that incorporates separation algorithms for these inequalities. Furthermore, we have proposed two matheuristic algorithms for the MDdGRP focused on trying to find better solutions by sequentially adding some promising intermediate points for drones to enter and exit the lines.

The results obtained from both matheuristic algorithms demonstrate the superiority of the algorithm that starts by generating solutions without intermediate points and subsequently adds some promising ones to the best solutions found. This approach outperforms the algorithm that first adds some intermediate points to the instance and then finds a set of initial solutions.

The branch-and-cut algorithm is capable of solving almost all the MDGRP(0) instances of small size but it struggles to solve the medium ones and cannot solve any of the large ones. By comparing the results of the branch-and-cut algorithm with those obtained with the first step of the best matheuristic (before including intermediate vertices), we have shown that the matheuristic is capable of obtaining optimal solutions in some instances. For those instances in which optimality is not proven, the gaps are significantly small. The inclusion of intermediate points, as evidenced by the solutions obtained by the matheuristic algorithm, allows us to obtain, within short computational times, better solutions than without them, even when using exact procedures.

As future lines of research, we want to consider an extension of the problem in which the number of depots is greater than the number of available drones and we have to choose which depots to use. This allows for considering a wide range of potential depots and selecting the optimal ones for launching the drones. Furthermore, we intend to study the situation in which each drone can perform an indefinite number of flights from its depot as well as considering different objectives.

Acknowledgments

This work was supported by grant PID2021-122344NB-I00 funded by MCIN/AEI/10.13039/501100011033 and "ERDF A way of making Europe."

References

Ahabchane, C., Langevin, A., M., T.,2020. The mixed capacitated general routing problem with time–dependent demands. *Networks* 76, 467–484.

Amorosi, L., Puerto, J., Valverde, C., 2021. Coordinating drones with mothership vehicles: the mothership and drone routing problem with graphs. *Computers & Operations Research* 136, 105445.

Amorosi, L., Puerto, J., Valverde, C., 2024. An extended model of coordination of an all-terrain vehicle and a multivisit drone. *International Transactions in Operational Research.* 31, 2, 780–806.

Archetti, C., Bertazzi, L., Laganà, D., Vocaturo, F., 2017. The undirected capacitated general routing problem with profits. *European Journal of Operational Research* 257, 3, 822–833.

Bach, L., Hasle, G., Wøhlk, S., 2013. A lower bound for the node, edge, and arc routing problem. *Computers & Operations Research* 40, 4, 943–952.

Barahona, F., Grotschel, M., 1986. On the cycle polytope of a binary matroid. *Journal of Combinatorial Theory B* 40, 1, 40–62.

Bosco, A., Laganà, D., Musmanno, R., Vocaturo, F., 2013. Modeling and solving the mixed capacitated general routing problem. *Optimization Letters* 7, 1451–2469.

Campbell, J.F., Corberán, Á., Plana, I., Sanchis, J.M., Segura, P., 2023. The multi-purpose *k*-drones general routing problem. *Networks* 82, 4, 437–458.

Campbell, J.F., Corberán, Á., Plana, I., Sanchis, J.M., 2018. Drone arc routing problems. *Networks* 72, 4, 543–559.

Campbell, J.F., Corberán, Á., Plana, I., Sanchis, J.M., Segura, P., 2021. Solving the length constrained *k*-drones rural postman problem. *European Journal of Operational Research* 292, 1, 60–72.

Campbell, J.F., Corberán, Á., Plana, I., Sanchis, J.M., Segura, P., 2022. Polyhedral analysis and a new algorithm for the length constrained *k*-drones rural postman problem. *Computational Optimization and Applications* 83, 67–109.

Chow, J.Y.J., 2016. Dynamic UAV-based traffic monitoring under uncertainty as a stochastic arc-inventory routing policy. *International Journal of Transportation Science and Technology* 5, 3, 167–185.

Corberán, Á., Plana, I., Rodríguez-Chía, A.M., Sanchis, J.M., 2013. A branch-and-cut algorithm for the maximum benefit Chinese postman problem. *Mathematical Programming* 141, 21–48.

Corberán, Á., Plana, I., Sanchis, J.M., 2007. A branch & cut algorithm for the windy general routing problem and special cases. *Networks* 49, 4, 245–257.

Corberán, Á., Plana, I., Sanchis, J.M., 2015. The windy general routing polyhedron: a global view of many known arc routing polyhedra. *SIAM Journal on Discrete Mathematics* 22, 2, 606–628.

Di Puglia Pugliese, L., Guerriero, F., 2017. Last-mile deliveries by using drones and classical vehicles. In Sforza, A., Sterle, C. (eds), *Optimization and Decision Science: Methodologies and Applications*, Springer International Publishing, Cham, pp. 557–565.

Ghiani, G., Laporte, G., 2000. A branch-and-cut algorithm for the undirected rural postman problem. *Mathematical Programming* 87, 467–481.

Hansen, P., Mladenović, N., 2001. Variable neighborhood search: principles and applications. *European Journal of Operational Research* 130, 3, 449–467.

Hasle, G., 2014. Arc routing applications in the newspaper industry. In Corberán, Á., Laporte, G. (eds), *Arc Routing: Problems, Methods, and Applications*. MOS–SIAM, Philadelphia, pp. 371–396.

Hierholzer, C., 1873. Uber die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen* 6, 30–32.

Kim, D., Moon, I., 2024. Scheduling-location problem with drones. *International Transactions in Operational Research*. doi: https://doi.org/10.1111/itor.13423.

Lenstra, J., Rinnooy Kan, A., 1976. On general routing problems. *Networks* 6, 273–280.

Li, M., Zhen, L., Wang, S., Lv, W., Qu, X., 2018. Unmanned aerial vehicle scheduling problem for traffic monitoring. *Computers & Industrial Engineering* 122, 15–23.

Luo, H., Zhang, P., Wang, J., Wang, G., Meng, F., 2019. Traffic patrolling routing problem with drones in an urban road system. *Sensors* 19, 23, 5164.

Macrina, G., Di Puglia Pugliese, L., Guerriero, F., Laporte, G., 2020. Drone-aided routing: a literature review. *Transportation Research Part C: Emerging Technologies* 120, 102762.

Montoya-Torres, J., López Franco, J., Nieto Isaza, S., Felizzola Jiménez, H., Herazo-Padilla, N., 2015. A literature review on the vehicle routing problem with multiple depots. *Computers & Industrial Engineering* 79, 115–129.

Murray, C.C., Chu, A.G., 2015. The flying sidekick traveling salesman problem: optimization of drone-assisted parcel delivery. *Transportation Research Part C: Emerging Technologies* 54, 86–109.

Orloff, C.S., 1974. A fundamental problem in vehicle routing. *Networks* 4, 1, 35–64.

Oruc, B., Kara, B., 2018. Post-disaster assessment routing problem. *Transportation Research Part B: Methodology* 116, 76–102.

Pandi, R., Muralidharan, B., 1995. A capacitated general routing problem on mixed networks. *Computers & Operations Research* 22, 5, 465–478.

Prins, C., Bouchenoua, S., 2005. A memetic algorithm solving the VRP, the carp and general routing problems with nodes, edges and arcs. In Hart, W.E., Smith, J.E., Krasnogor, N. (eds), *Recent Advances in Memetic Algorithms*, Studies in Fuzziness and Soft Computing, Vol. 166. Springer, Berlin,, pp. 65–85.

Rojas Viloria, D., Solano-Charris, E.L., Muñoz Villamizar, A., Montoya-Torres, J.R., 2021. Unmanned aerial vehicles/drones in vehicle routing problems: a literature review. *International Transactions in Operational Research* 28, 4, 1626–1657.

Salazar-Aguilar, M., Langevin, A., Laporte, G., 2013. The synchronized arc and node routing problem: application to road marking. *Computers & Operations Research* 40, 1708–1715.

Wang, Z., Sheu, J.B., 2019. Vehicle routing problem with drones. *Transportation Research Part B: Methodological* 122, 350–364.