European Journal of Operational Research 292 (2021) 60–72

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

Solving the length constrained *K*-drones rural postman problem

James F. Campbell[a], Ángel Corberán[b], Isaac Plana[c,*], José M. Sanchis[d], Paula Segura[d]

[a] Supply Chain and Analytics Department, University of Missouri-St. Louis, St. Louis, MO 63121-4499, USA
[b] Department d'Estadística i Investigació Operativa, Universitat de València, Avda. Dr. Moliner 50, Burjassot, Valencia 46100, Spain
[c] Department de Matemáticas Para la Economía y la Empresa, Universitat de València, Avda. Tarongers s/n, Valencia 46022, Spain
[d] Department de Matemática Aplicada, Universidad Politécnica de Valencia, Camino de Vera s/n, Valencia 46022, Spain

ARTICLE INFO

Article history:
Received 15 January 2020
Accepted 27 October 2020
Available online 1 November 2020

Keywords:
Logistics
Drones
Arc routing
Length constraints
Matheuristic

ABSTRACT

In this paper we address the Length Constrained *K*-Drones Rural Postman Problem (LC *K*-DRPP). This is a continuous optimization problem where a fleet of homogeneous drones have to jointly service (traverse) a set of (curved or straight) lines of a network. Unlike the vehicles in classical arc routing problems, a drone can enter a line through any of its points, service a portion of that line, exit through another of its points, then travel directly to any point on another line, and so on. Moreover, since the range of the drones is restricted, the length of each route is limited by a maximum distance. Some applications for drone arc routing problems include inspection of pipelines, railway or power transmission lines and traffic monitoring.

To deal with this problem, LC *K*-DRPP instances are digitized by approximating each line by a polygonal chain with a finite number of points and allowing drones to enter and exit each line only at these points. In this way we obtain an instance of the Length Constrained *K*-vehicles Rural Postman Problem (LC *K*-RPP). If the number of points used to discretize the lines is large, the LC *K*-RPP instance can be extremely large and, hence, very difficult to solve optimally. Even heuristic algorithms can fail in providing feasible solutions in reasonable computing times. An alternative is to generate smaller LC *K*-RPP instances by approximating each line with few but "significant" segments.

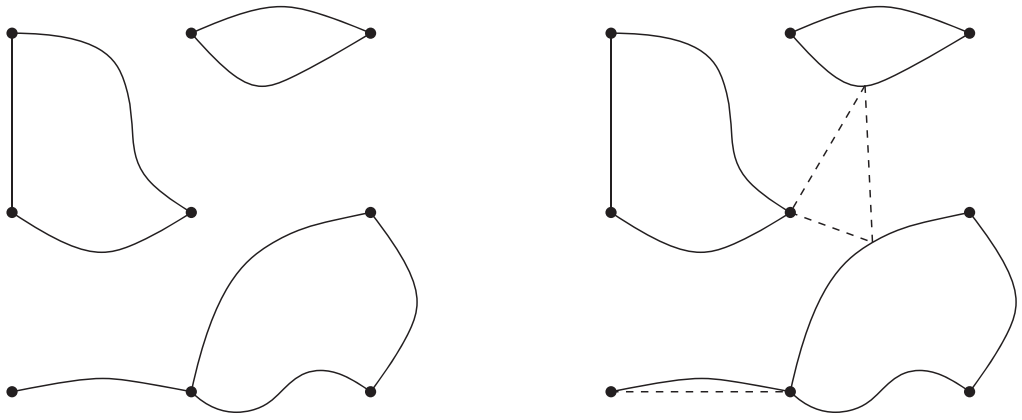
We present a formulation and some valid inequalities for the LC *K*-RPP. Based on this, we have designed and implemented a branch-and-cut algorithm for its solution. Moreover, in order to be capable of providing good solutions for large LC *K*-RPP instances, we propose a matheuristic algorithm that begins by finding good solutions for the LC *K*-RPP instance obtained by approximating each line by a single segment. Then, to find better solutions, some promising intermediate points are sequentially incorporated. Extensive computational experiments to assess the performance of both algorithms are performed on several sets of instances from the literature.

1. Introduction

Classical arc routing problems (ARPs) consist of finding a tour, or a set of tours, with total minimum cost traversing (servicing) a set of links (arc or edges), called required links, of a graph (see Corberán, Eglese, Hasle, Plana, & Sanchis, 2020; Corberán & Laporte, 2014; Mourão & Pinto, 2017). Well-known ARPs are the Chinese postman problem (CPP) and the rural postman problem (RPP), where a single vehicle has to traverse all or some of the links of the graph, respectively, and the capacitated arc routing problem (CARP), where there is a fleet of vehicles with limited capacity to jointly service the required links. In classical ARPs, the streets to be cleaned, roads where snow must be removed, or pipelines to be inspected, for example, are represented by edges or arcs of a network that ignore the line shape (although not its true cost or distance) since the vehicles have to traverse an arc from one endpoint to the other one. Further, the vehicles in these classical ARPs are not permitted to travel off the network. In the following, we call these Postman ARPs, and we use Drone ARPs to refer to arc routing problems where aerial drones are used to optimize the service of the required links of a graph, and these drones have the capability to travel off the network. Aerial drones may be used to replace ground vehicles because of their reduced costs, higher speeds and/or safety improvements.

* Corresponding author.
E-mail addresses: campbell@umsl.edu (J.F. Campbell), angel.corberan@uv.es (Á. Corberán), isaac.plana@uv.es (I. Plana), jmsanchis@mat.upv.es (J.M. Sanchis), psegmar@upvnet.upv.es (P. Segura).

https://doi.org/10.1016/j.ejor.2020.10.035
0377-2217/



(a) Required lines (b) Route of a single drone

Fig. 1. The shape of the lines is important when using drones.

As pointed out in Campbell, Corberán, Plana, and Sanchis (2018), the use of drones to perform the service in ARPs involves significant changes in the traditional way of modeling and solving these problems. These changes are mainly due to the ability of drones to travel directly between two points, not necessarily between vertices of the graph. Hence, drones do not need to follow the edges of the graph and may start the service of an edge at any point along the edge. On the one hand, this means it is important to take into account the shape of the lines to service, as the example in Fig. 1 illustrates. Fig. 1b shows the optimal solution of the Drone RPP instance depicted in Fig. 1a, where dotted edges represent the movement of the drone outside the graph, and its cost is the sum of the costs of the required lines (a fixed value) plus the deadheading cost, which is the one to minimize in the Drone RPP). On the other hand, an important consequence of taking the shape of the lines into consideration is that the problem becomes a continuous optimization problem with an infinite and non-countable number of feasible solutions. In Campbell et al. (2018) this drawback is overcome by modeling the curved lines by means of polygonal chains and then allowing the drones to enter and leave each curve only at the points of the polygonal chain. In particular, a curve can be approximated by its endpoints and a set of intermediate points defining segments with an associated service cost so that the sum of the costs associated with all the segments of the polygonal chain equals the service cost of the original curve. Hence, the segments define the set of required edges of the transformed graph and, since the drones can travel directly between any two points, the set of (non-required) edges form a complete graph with (deadheading) costs given by the Euclidean distances.

Although there are a large number of articles in the academic literature that address node routing problems with drones (see Chung, Sah, & Lee, 2020; Khoufi, Laouiti, & Adjih, 2019; Macrina, Di Puglia Pugliese, Guerriero, & Laporte, 2020; Otto, Agatz, Campbell, Golden, & Pesch, 2018; Poikonen & Campbell, 2020), there are only a few papers dealing with the use of aerial drones for ARPs. Oh, Kim, Tsourdos, and White (2014); Oh, Shin, Tsourdos, White, and Silson (2011) consider a search problem using drones on an unconnected portion of a road network, and present a greedy insertion heuristic algorithm. Dille and Singh (2013) consider a related road (edge) covering problem for a drone carrying sensor with a given radius of coverage. They convert this to a TSP by discretizing the road network into a set of points where each point covers a certain part of the road network based on the sensor range of a drone. Results were compared to those in Oh et al. (2014) for several simulated networks. Chow (2016) considers the use of drones to monitor road traffic and presents deterministic and stochastic multi-period arc-inventory routing problems (AIRP). Li, Zhen, Wang, Lv, and Qu (2018) also consider inventory routing with a capacitated ARP (CARP) for monitoring road traffic, where over-monitoring of an edge is treated as inventory and is penalized. A mixed integer programing model is used to solve small problems with up to 40 links, and a local branching algorithm is used to solve larger problems with up to 50 links. A cooperative system of a ground vehicle and a drone for the inspection of powerlines is presented in Liu, Shi, Liu, Huang, and Zhou (2019). The ground vehicle is used as a mobile base station that can drive to several locations near the powerline in the road network to launch the drone and recharge its battery. Several heuristics are proposed to design the routing planning for the vehicle and the drone to minimize the completion time or the cost for inspecting all the powerlines. The performance of the heuristics is illustrated in three practical instances based on the powerline and road network of Ji (China). Another combined problem of a ground vehicle and a drone is studied by Luo, Zhang, Wang, Wang, and Meng (2019). It concerns the road traffic patrolling problem in an urban road system, where the vehicle transports the drone and performs traffic patrols through the release and recovery of the drone, while the vehicle also performs mission visits. The authors present two heuristics for this problem and a case study on the relevant roads in Hefei (China).

Applications for drone arc routing problems include inspection of railroad and energy infrastructure, and detection or surveillance of borders. While early drone use has often been limited to short drone flights due to line-of-sight regulations and small battery capacities for electric drones, regulatory changes and newer drones allow long flights and BVLOS (beyond visual line-of-sight) flight. For example, the major US railroad BNSF has flown BVLOS flights (with an exemption from FAA regulations) since 2016, with the first flight being for a 132-mile stretch of track Franz (2018). Other major railroads around the world now operate drone fleets for infrastructure and asset inspection, vegetation monitoring, etc. United Nations ESCAP (2019). Drones used for long range (or long endurance) applications (as for BNSF) are typically large aircraft (e.g., wingspans of 12–15 feet), often with both electric and gas motors (electric motors allow VTOL (vertical take-off and landing) and gas engines allow efficient forward flight) (e.g., Franz, 2018; Railway, 2019). For example, BNSF reports flights of up to 16 hours (Railway, 2019). Reports of drones used for the energy industry include BVLOS applications for inspection of power transmission lines (e.g., 90 kilometers in Sweden, Press, 2020), up to 150 miles of lines in the US (Garrett-Glaser, 2019; Zeisee, 2019), and for gas and electricity lines in the UK (Networks, 2019). Other applications

with drone flights covering linear features include surveillance along borders (Delair, 2020) and surveillance to prevent marine ingress near nuclear power plants (Catapult, 2020). Drones have been used for asset inspection, in which case the area to be inspected can be represented as a network of linear features (providing sensor coverage of the desired areas). These applications cover a diverse set of assets in a wide range of environments, including bridges, oil rigs (above and below the sea surface), and cargo ships (both the inside and outside of the hull, including inspection below the sea surface) (Bonnin-Pascuala & Ortiz, 2019; Knight, 2019; Seo, Dudue, & Wacker, 2018).

The recent paper devoted to Drone ARPs (Campbell et al., 2018) discuses the relationship between Drone and Postman ARPs. In particular, it is shown that the worst-case ratio of the costs of Postman ARPs with respect to that of Drone ARPs is infinite. The authors define a Drone ARP for a single vehicle, the Drone RPP, for which they propose an algorithm that iteratively solves RPP instances with an increasing number of intermediate points. This procedure is capable of solving medium size instances of this very hard problem. In addition, the length constrained K-drones rural postman problem (LC K-DRPP) is introduced and some of its characteristics are described.

This paper is organized as follows. In Section 2 we present the LC K-DRPP and in Section 3 we propose a formulation with binary variables and some families of valid inequalities for the LC K-RPP. The general scheme of the matheuristic we propose for this problem is described in Section 4, while Section 5 presents a branch-and-cut algorithm (B&C) based on the proposed formulation. The computational results obtained with the B&C and the matheuristic on a large set of instances are shown and commented in Section 6. Finally, some conclusions are drawn in Section 7.

2. The Length Constrained K-Drones Rural Postman Problem

The Length Constrained K-Drones Rural Postman Problem (LC K-DRPP) was introduced in Campbell et al. (2018) and defined as follows: Given a set of lines, each one with an associated service cost, and a point called the depot, assuming that the cost of deadheading between any two points is the Euclidean distance, and given a constant L, find a set of drone routes starting and ending at the depot and with lengths no greater than L such that they jointly traverse all the given lines completely with minimum total cost. This problem envisions a set of K drones, each with route length limit L, that start and end at a common (given) depot and need to be routed to cover a set of lines at minimum cost, where the drones can travel in a straight line between any two points.

Unlike the vehicles in classical ARPs, which have to follow the links of a given graph, drones can fly directly between any two points. Thus, a drone can enter a line that requires service through any of its (infinite) points, traverse and service part of it, exit the line through another of its points, then travel directly to any point on another required line, and so on. In this way, shorter solutions can be obtained using drones than with ground vehicles. The price to pay is that the problem is much more difficult. In fact, it is a continuous optimization problem, with an uncountable number of feasible solutions.

To deal with this problem, each specific instance can be digitized, defining each curved line by a (usually) large set of points, each one with its corresponding coordinates. In other words, each line is approximated by a polygonal chain and drones are allowed to enter and leave each line only at the points of the polygonal chain, thus obtaining a discrete optimization problem. Obviously, the greater the number of points, the closer the discrete problem is to the continuous problem.

When an LC K-DRPP instance is discretized, we obtain an instance of a Postman ARP, the Length Constrained K-vehicles Rural Postman Problem (LC K-RPP). These last instances are defined on an undirected graph $G = (V, E)$, where the set of vertices V is formed by all the points of the polygonal chains plus the depot, the set of required edges, $E_R \subset E$, is formed by all the segments of the polygonal chains, and the non-required edges, $E_{NR} \subset E$, define a complete graph over the vertex set V. The cost of traversing and servicing each required edge (a segment) $e \in E_R$, $c^s_e \geq 0$, is equal to the proportional part of the total cost of servicing the corresponding original line (assuming that the cost rate of servicing a line is the same in all its parts), while the (deadheading) cost, $c_e \geq 0$, associated with the traversal of a non-required edge $e = (i, j) \in E_{NR}$ is given by the Euclidean distance from i to j. Note that, for each required edge $e \in E_R$, there is a non-required parallel edge $e' \in E_{NR}$. We assume $c^s_e \geq c_{e'}$. The goal of the LC K-RPP is to find K routes (tours) starting and ending at the depot that jointly traverse all the required edges and such that the total cost, or length, of each route does not exceed a maximum value L, with minimum total cost.

Note that, if the number of points used to discretize the lines of an LC K-DRPP instance is large, the corresponding LC K-RPP instance can be extremely large and therefore very difficult to solve optimally, and even heuristic algorithms can fail to provide feasible solutions in reasonable computing times. For example, an instance with 84 polygonal chains with 20 intermediate points per chain has 1763 vertices, 1764 required edges, and 1,553,203 non-required edges. Moreover, unlike in the (single) Drone RPP, we cannot remove some non-required edges without losing some optimal solutions. An alternative is to generate smaller LC K-RPP instances by approximating each line with very few segments, provided that the intermediate points are significant points.

The smallest LC K-RPP instance, and the least tight approximation to the corresponding LC K-DRPP instance, is the one obtained by approximating each line of the LC K-DRPP instance by a single edge (a polygonal chain with a single segment, without intermediate points). Let us call these instances LC K-RPP(0). They can be meaningful in real situations where it is mandatory, or recommended, that each line is fully serviced by the same drone, from one endpoint to the other. We want to point out that since drones can fly directly between any two endpoints, in these instances the non-required edges form a complete graph, whereas in Postman ARPs the graph often corresponds to a sparse network.

3. A formulation for the LC K-RPP

We are going to formulate the LC K-RPP on the graph $G = (V, E)$ described in Section 2. This is an undirected multigraph with a non-required edge parallel to each required edge, with different costs c_e and c^s_e. Given that G is undirected, we note that the single tour associated with any drone traverses each edge in G at most twice (if a tour traverses an edge e three or more times, we can remove two traversals of e and obtain another tour with lower cost). Furthermore, it can be seen that, for each tour that traverses an edge e twice in the same direction, we can build an equivalent tour (with the same cost) that traverses e once in each direction. Therefore, we can assume that the tours associated with each drone traverse each edge at most once in each direction. We can formulate the LC K-RPP with the following binary variables.

For each edge $e = (i, j) \in E$ and for each drone $k \in \{1, \ldots, K\}$ we define two binary variables x^k_{ij}, x^k_{ji}. If e is required, variables x^k_{ij}, x^k_{ji} take the value 1 if, and only if, e is serviced and traversed by drone k from i to j or from j to i, respectively. If e is non-required, variables x^k_{ij}, x^k_{ji} take the value 1 if, and only if, e is deadheaded (traversed without service) by drone k from i to j or from j to i, respectively.

We use the following notation. Given a subset $S \subseteq V$, $\delta(S)$ denotes the edge set with one endpoint in S and the other one in

$V\setminus S$, and $E(S)$ denotes the set of edges with both endpoints in S. We denote $\delta_R(S)=\delta(S)\cap E_R$ and $E_R(S)=E(S)\cap E_R$. Finally, for any subset $F\subseteq E$, we denote $x^k(F)=\sum_{e=(i,j)\in F}(x^k_{ij}+x^k_{ji})$. The LC K-RPP can be formulated as follows:

$$\text{Minimize} \sum_{k=1}^{K}\sum_{e=(i,j)\in E_{NR}} c_e(x^k_{ij}+x^k_{ji}) + \sum_{k=1}^{K}\sum_{e=(i,j)\in E_R} c^s_e(x^k_{ij}+x^k_{ji}) \quad (1)$$

s.t.:

$$\sum_{(i,j)\in\delta(i)}(x^k_{ij}-x^k_{ji})=0,\ \ \forall i\in V,\ \ \forall k\in\{1,\dots,K\} \quad (2)$$

$$x^k(\delta(S))\geq 2(x^k_{ij}+x^k_{ji}),\ \ \forall S\subset V\setminus\{1\},\ \ \forall (i,j)\in E_R(S),\ \ \forall k \quad (3)$$

$$\sum_{k=1}^{K}(x^k_{ij}+x^k_{ji})=1,\ \ \forall (i,j)\in E_R \quad (4)$$

$$\sum_{e=(i,j)\in E_{NR}} c_e(x^k_{ij}+x^k_{ji})+\sum_{e=(i,j)\in E_R} c^s_e(x^k_{ij}+x^k_{ji})\leq L,\ \ \forall k\in\{1,\dots,K\} \quad (5)$$

$$x^k_{ij},x^k_{ji}\in\{0,1\},\ \ \forall (i,j)\in E,\ \ \ \forall k\in\{1,\dots,K\} \quad (6)$$

The objective function (1) minimizes the total cost of the routes. The first term represents the "deadheading cost" while the second represents the cost of the required edges that, due to constraints (4), is a constant and therefore can be removed:

$$\sum_{k=1}^{K}\sum_{e=(i,j)\in E_R} c^s_e(x^k_{ij}+x^k_{ji})=\sum_{e=(i,j)\in E_R} c^s_e\left(\sum_{k=1}^{K}(x^k_{ij}+x^k_{ji})\right)=\sum_{e=(i,j)\in E_R} c^s_e.$$

Symmetry constraints (2) force each drone k to exit a vertex i as many times as it enters it. Connectivity inequalities (3) ensure each single route is connected and connected to the depot, while constraints (5) guarantee that the length or cost of each route does not exceed L. The traversal of all the required edges exactly once is ensured by Eq. (4). Constraints (6) are the binary conditions for the variables.

The above formulation can be strengthened with the following inequalities. Given $e=(i,j)\in E_R$ and its corresponding parallel edge $e'=(i,j)'\in E_{NR}$, and given that the tour performed by a vehicle k travels at most once from i to j, the following *single-traversal* inequalities are satisfied:

$$x^k_{ij}+x^k_{(ij)'}\leq 1. \quad (7)$$

Moreover, *parity inequalities* are also valid and very useful to cut many fractional "solutions":

$$x^k(\delta(S)\setminus F)\geq x^k(F)-|F|+1, \forall S\subset V,\ \ \forall F\subseteq\delta(S)\ \textit{with}\ |F|\ \textit{odd},\ \ \forall k\in\{1,\dots,K\} \quad (8)$$

These inequalities are based on the fact that all vehicles have to traverse any edge cutset an even, or zero, number of times, and it is easy to see that they are valid for the K-RPP on G. Since $|F|$ is odd, the tours x^k for which $x^k(F)=|F|$ holds, should satisfy $x^k(\delta(S)\setminus F)\geq 1$. Tours x^k for which $x^k(F)<|F|$, obviously satisfy $x^k(\delta(S)\setminus F)\geq 0$. Inequalities (8) are referred to as cocircuit inequalities by Barahona and Grötschel (1986) and were proposed for the RPP by Ghiani and Laporte (2000).

4. A matheuristic for the LC *K*-DRPP

In this section we present a matheuristic algorithm for the LC K-DRPP. It begins by finding good solutions for the LC K-RPP(0) instances. Then we sequentially incorporate some promising intermediate points to find better solutions. Eventually, the matheuristic provides feasible solutions for the LC K-DRPP with the characteristic, typical of drones, that some lines are serviced in different parts by different drones or, equivalently, that some drones only service part of some lines. The procedure developed here has three phases.

- In phase 1, we consider the LC K-RPP(0) instance in which each original line is approximated by only one (required) edge without intermediate points. The algorithm first computes a "giant tour" traversing all the required edges by optimally solving an RPP on the corresponding graph G. This giant tour is then partitioned into K routes, one for each drone, to obtain an LC K-RPP solution. The process of forming a giant route and partitioning it into K routes is repeated several times to obtain different LC K-RPP solutions. This is described in Section 4.1. Three local search procedures are then applied to each of these solutions to improve the routes (described in Section 4.2), and then each of the single routes of the resulting solutions are optimized (described in Section 4.3).
- In phase 2, we consider the n best solutions obtained in phase 1. For each of these solutions, we add an intermediate vertex to each required edge, thus obtaining n solutions of the LC K-RPP instance where each original line is approximated by a polygonal chain with two segments (edges). We then apply the local search and the single route optimization procedures to each of these solutions to possibly improve them. We call this procedure "1-splitting" and it is described in Section 4.4.
- In phase 3, the most "promising" segments of each solution obtained in phase 2 are split again by adding one new intermediate vertex to them, while some unused intermediate vertices are removed. This procedure is called "3-splitting" because some of the original lines are approximated by a polygonal chain with three intermediate vertices (four edges). Again, the local search and the single route optimization procedures are applied to improve each solution. In a second step of this phase, called "7-splitting", we add once again new intermediate vertices on the "promising" segments and remove some unused vertices of each improved solution. After applying the local search and the single route optimization procedures to the resulting solutions, we keep the best of them. This is described in Section 4.5.

Before describing in detail the different procedures that compose the matheuristic algorithm, let us introduce some notation. A *solution* S is a set of K drone routes, with each route starting and ending at the depot and of length no greater than L, such that each required edge is serviced (traversed) in exactly one route. A route T is represented by a sequence $\{(i,j),(k,l),\dots,(u,v)\}$ of required edges, which will be denoted E_T. A route T is called *feasible* if its length is not greater than L. It is assumed that route T services the required edges on E_T in the same order as they appear in the sequence. It is also assumed that the deadheading from the depot to vertex i, from the end of a required edge to the beginning of the following edge in the sequence, and from vertex v back to the depot, is done by traversing the corresponding non-required edge. Note that the length of a route T is the sum of the costs of the required edges on E_T and the deadheading costs of the non-required edges. Throughout this section we will denote by T_i the ith route in S, with $i\in\{1,2,\dots,K\}$.

4.1. Initial LC K-DRPP solutions

In the first step of the algorithm, a giant tour T_G on $G=(V,E)$ (the graph corresponding to the LC K-RPP(0) instance) traversing all the required edges is found by solving this RPP instance optimally with the branch-and-cut algorithm proposed by Corberán, Plana, and Sanchis (2007). This giant tour (see Fig. 2a) is then partitioned into K routes of length no greater than L (see Fig. 2b) by means of the procedure proposed by Ulusoy (1985) for the

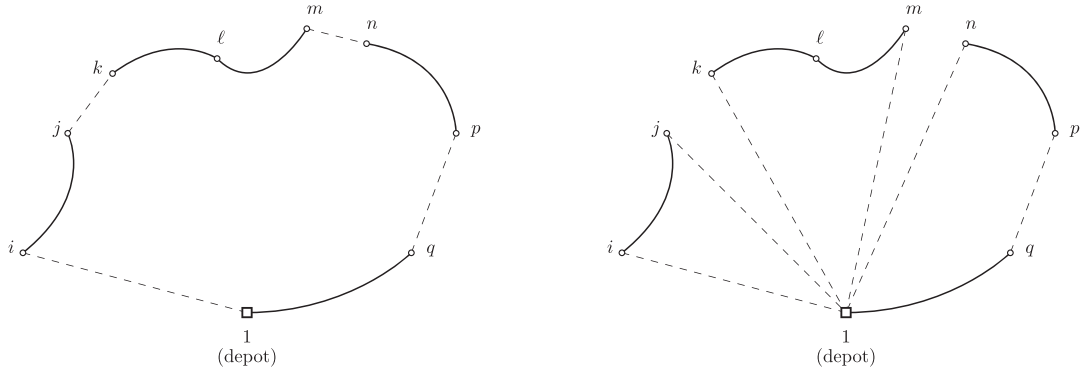


Fig. 2. Splitting an optimal giant tour T_G into K feasible routes.

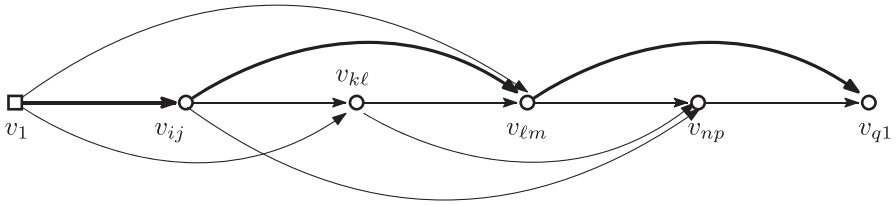


Fig. 3. The auxiliary graph G^* generated from $T_G = \{(i,j),(k,\ell),(\ell,m),(n,p),(q,1)\}$.

CARP. This procedure works on an auxiliary directed graph G^* constructed from T_G as follows:

(i) G^* is a directed graph with $|E_R|+1$ nodes that admits a rectilinear representation (see Fig. 3). The first node of G^*, denoted by v_1, corresponds to the depot of G. Associated with each required edge (i,j) of G we add a node v_{ij} on G^*. The nodes are arranged from left to right following the order in which their associated required edges are traversed in the giant tour T_G. Recall that each required edge is traversed only once in the optimal giant tour since drones can travel between any pair of vertices of G through a non-required edge with equal or lower cost due to the completeness of the graph (V, E_{NR}).

(ii) Each arc on graph G^* represents a feasible drone route on G. An arc from node v_{ij} to node $v_{k\ell}$ is added to G^* if the required edge (i,j) is the last one serviced before the edge (k,ℓ) in the giant tour T_G. This arc represents the route starting at the depot, deadheading to vertex k, servicing edge (k,ℓ), and deadheading back to the depot, that is, the route $\{(k,\ell)\}$. The cost (length) of arc $(v_{ij}, v_{k\ell})$ in G^* is equal to the cost of its associated route. Furthermore, an arc from node v_1 to the first node v_{ij} is added to G^* with cost that of the route $\{(i,j)\}$ in G (see Fig. 3).

(iii) There are additional arcs included in graph G^* associated with feasible drone routes that service more than one required edge. An arc (v_{rs}, v_{tw}) is added to the auxiliary graph if the route from the first required edge after (r,s) to edge (t,w) is feasible. Furthermore, there is an arc in G^* from node v_1 to the node v_{tw} if the route from the first required edge of the giant tour to edge (t,w) is feasible. The length of these arcs is the cost associated with the corresponding route. In Fig. 3, arc (v_{ij}, v_{np}) in G^* implies that the route $\{(k,\ell),(\ell,m),(n,p)\}$ has a length no greater than L. However, arc (v_1, v_{np}) is not included in G^* because the route $\{(i,j),(k,\ell),(\ell,m),(n,p)\}$ is not feasible.

In graph G^* we compute a shortest path from node v_1 to node v_{q1} using the topological ordering, a technique that calculates shortest paths from a single source in $\mathcal{O}(|V|+|E|)$ time for directed acyclic graphs (see Cormen, Leiserson, Rivest, & Stein, 2009). In Ulusoy (1985) it is proved that the set of arcs in this shortest path defines a partition of the giant tour T_G into K feasible tours, and this partition is optimal regarding the ordering of the traversal of the required edges in T_G. For example, the shortest path in the graph G^* in Fig. 3, represented in bold lines, corresponds to the K feasible routes depicted in Fig. 2b. These K routes define a solution S of the LC K-RPP(0) to which we will apply the local search phase.

In order to obtain a larger set of initial solutions, we repeat the above algorithm with other giant tours on G defined by other Eulerian circuits obtained from the optimal RPP solution. Note that an RPP solution is an Eulerian graph that can be traversed in different ways. We try to generate different Eulerian circuits by applying the Hierholzer algorithm (Hierholzer, 1873) $|E_R|$ times, starting each time with a different required edge. We thus obtain a set $\bar{S}$ of different initial LC K-RPP(0) solutions, with $|\bar{S}| \le |E_R|$.

4.2. Local search procedures

Three local search procedures are used in the matheuristic to attempt to improve a set of drone routes. They are based on the exchange of edges between different routes in order to minimize the total cost. The first two procedures explained below are applied by following a *first improvement* strategy. These procedures are applied to the initial solutions for the LC K-RPP(0) in $\bar{S}$. They will also be applied later to the set of solutions in the 1-splitting, 3-splitting and 7-splitting procedures.

4.2.1. 0 to ℓ – exchange

In this procedure, a move consists of removing ℓ consecutive required edges from the route servicing them and inserting all of them between two consecutive required edges of another route. The algorithm uses the following strategy. We consider the removal of all the possible sets of ℓ consecutive required edges and their insertion in all the possible positions of other routes such that the total cost is smaller than the original and the length of the new route does not exceed L. The procedure starts with $\ell = 1$.

If no exchange that improves the current solution is found, we increment ℓ by 1, otherwise ℓ is reset to 1. The procedure stops when $\ell = \ell_{max}$ and there are no moves (insertions) that improve the total cost, where ℓ_{max} is a given parameter.

4.2.2. ℓ_1 to ℓ_2 – exchange

This procedure is similar to the one described above but now a move consists of interchanging ℓ_1 consecutive required edges from a route T_i with ℓ_2 consecutive required edges from another route T_j, with $\ell_1 \leq \ell_2$ and $i, j \in \{1, 2, \ldots, K\}$. The algorithm starts with $\ell_1 = 1$ and $\ell_2 = 1$, and tries to interchange the required edges between the two routes in order to find an improving move. If there are no exchanges that reduce the total cost, then ℓ_2 increases by one unit and the process is repeated. If ℓ_2 reaches ℓ_{max} and no improving exchanges are found, then ℓ_1 increases by one unit and ℓ_2 is set equal to ℓ_1. If an improving exchange is discovered, it is executed and ℓ_1, ℓ_2 are reset to 1. The algorithm ends when $\ell_1 = \ell_2 = \ell_{max}$ and there are no exchanges that improve the total cost.

4.2.3. Destroy and repair

In each iteration of the destroy and repair algorithm, we randomly choose r required edges, where r is a random value between 2 and 8, and these edges are removed from the routes servicing them. This strategy randomly shortens some of the routes of the solution in order to possibly complete them again differently with lower total cost. Then, we try to relocate these required edges one by one in the same order they were removed. Each required edge is inserted in the route and the position that minimizes the total cost, only if the length of the resulting route does not exceed L. Note that it is possible that a required edge can not be placed in its original position because another edge has been previously added to its route. If an edge cannot be inserted in any route, a new route servicing it is created. If the obtained solution does not improve the starting one, the changes made in this iteration are discarded. This procedure is repeated until r_{max} consecutive iterations without any improvement are performed, where r_{max} is a given parameter.

4.3. Optimization of the routes

The route optimization phase is applied to a solution S once the local search procedures are terminated. It is aimed at optimizing each drone route in S by solving an RPP instance with the branch-and-cut algorithm proposed in Corberán et al. (2007). Thus, for each route T_i in the solution S we define an RPP instance on graph G with set of required edges E_{T_i} formed by the required edges that are traversed and serviced in route T_i. This RPP instance is then solved optimally. Each obtained route is feasible and has a cost less than or equal to that of the original route.

4.4. 1-splitting procedure

The best solutions obtained after applying the local search and the route optimization procedures to all the initial solutions in $\bar{S}$ are stored and define the set $\bar{S}_b$. In order to improve them, we first apply the 1-splitting procedure to each $S \in \bar{S}_b$ as follows. An intermediate vertex (equi-distant from both endpoints) is added to each required edge to obtain a solution S' of the LC K-RPP with twice the number of original required edges. Thus, a required edge (i, j) of S is transformed in two required edges $(i, i_1), (i_1, j)$, where i_1 denotes the intermediate point added, that will be traversed consecutively in the "new" solution S'. We then apply the local search and the route optimization procedures to S' to try to obtain a better solution. Note that with this splitting procedure we are allowing the drone to enter and leave any edge through its middle point, making it possible to obtain a solution better than the starting one (prior to the 1-splitting).

For a better understanding of this phase, Fig. 4 shows an example of the behavior of a solution before (Fig. 4a) and after (Fig. 4b) the 1-splitting procedure. For this instance, the deadheading cost is reduced by 6.12% due to drones entering and leaving three of the original required edges through their middle point.

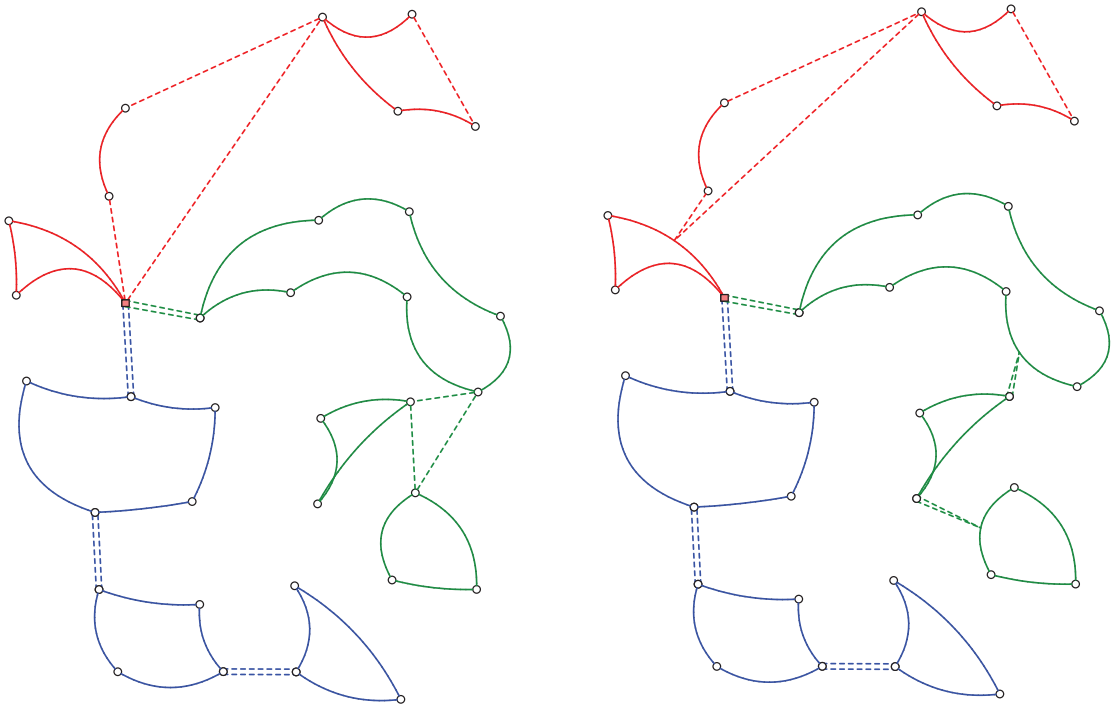
4.5. 3-splitting and 7-splitting procedures

The last phase of the matheuristic focuses on improving the intensification of the search for good solutions. The idea is to generate a new set of intermediate vertices for each solution considered (vertices that a drone can use to enter and leave an edge) based on the behavior of the drones on each route of the solution. Let $\bar{S}'_b$ denote the set of solutions obtained after applying the 1-splitting phase to each solution in $\bar{S}_b$. The K routes of any solution $S' \in \bar{S}'_b$ service $2|E_R|$ required edges. The idea now is to split again these required edges to give more options to the drones in order to possibly shorten their routes. However, splitting all $2|E_R|$ required edges would lead to apply the local search and route optimizing algorithms to a solution with $3|E_R| + |V|$ vertices, $4|E_R|$ required edges and $(3|E_R| + |V|) \times (3|E_R| + |V| - 1)/2$ non-required edges, which would be an excessive computational effort, more so if we consider that most of the non-required edges have a very low probability of being used by the drones.

Hence, instead of splitting all the $2|E_R|$ required edges, in each solution we split only those required edges incident to a non-required edge, as this creates new promising locations for the drone to enter or leave a required edge. Thereby, for each solution $S' \in \bar{S}'_b$, the required edges of S' that are incident with a non-required edge used by any of the routes are split by introducing a new intermediate vertex on it. Furthermore, in order to reduce the computational effort of this phase, we remove the intermediate vertices added in the 1-splitting procedure that are not incident with non-required edges in the solution (as these may be unlikely to be used in a near-optimal solution). We call this "3-splitting" because some of the original lines have been approximated by a polygonal chain with three intermediate vertices (see edge (i, j) in Fig. 5b).

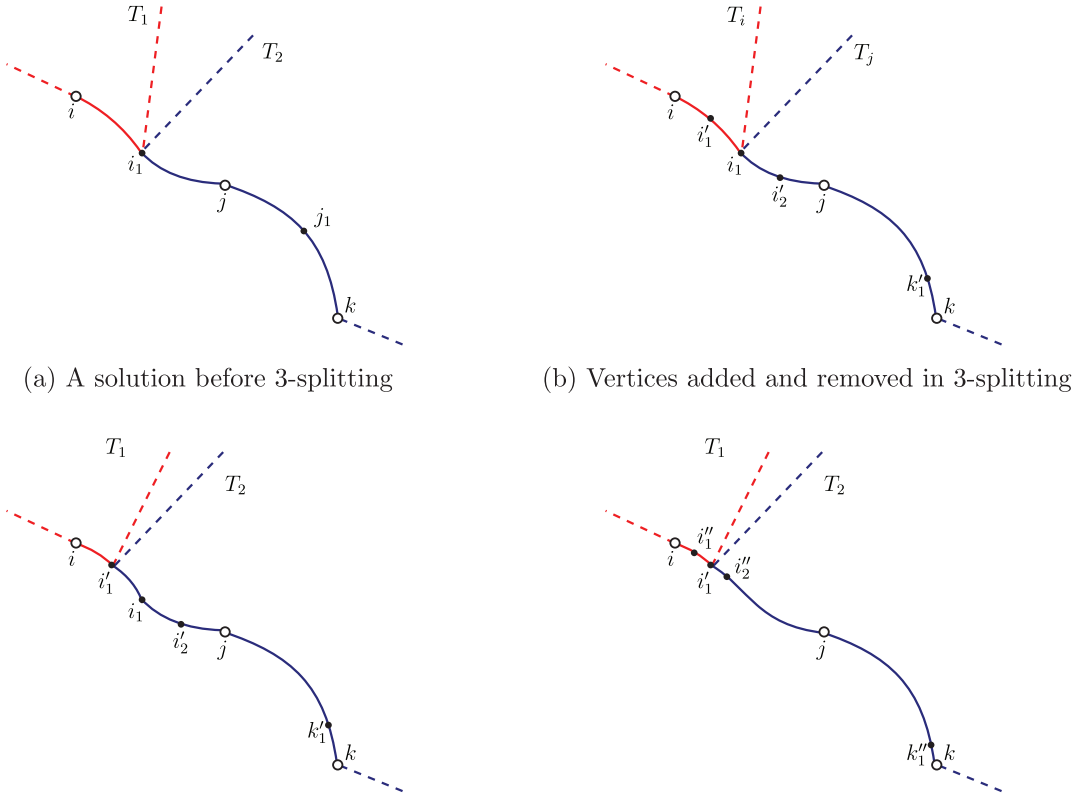
Fig. 5a and 5b illustrate how the 3-splitting works in a part of a solution where two routes, T_1 and T_2, are involved. On these figures, white nodes i, j, and k represent original vertices, whereas black nodes are intermediate vertices added on the required edges (in 1-splitting). Dashed lines represent (incomplete) non-required edges. The 3-splitting procedure (Fig. 5b) adds vertices i'_1 and i'_2 in the middle of the edges (i, i_1) and (i_1, j), respectively, because vertex i_1 (added in the 1-splitting phase) is incident with non-required edges. Moreover, vertex k'_1 is added to edge (j_1, k) and vertex j_1 (added in the 1-splitting phase) is removed. Then we apply the local search and the route optimization procedures. The result is shown in Fig. 5c, where the drone entry/exit of edge (i, j) shifts from i_1 to i'_1.

The 7-splitting phase is the last part of the algorithm and it works similarly to the 3-splitting. Again the idea is to add new intermediate vertices "near" to those that are incident with non-required edges in the solution and remove the intermediate vertices previously added that are not incident with non-required edges in the solution. The intermediate vertices added are selected among the seven vertices obtained when the original line is partitioned in eight segments of similar length. Fig. 5c and 5d show how the 7-splitting works. Fig. 5d is created by adding two 7-splitting nodes i''_1, i''_2 and removing two nodes i_1 (added in 1-splitting) and i'_2 (added in 3-splitting); and also adding k''_1 and removing k'_1. Local search and the route optimization algorithms are



(a) Deadheading cost: 1808.24 (b) Deadheading cost: 1697.51

Fig. 4. Two solutions of a DroneRPP68 instance before and after the 1-splitting procedure.



(a) A solution before 3-splitting (b) Vertices added and removed in 3-splitting

(c) A solution before 7-splitting (d) Vertices added and removed in 7-splitting

Fig. 5. Illustration of 3- and 7-splitting.

applied and the best solution among the $|\bar{S}_b|$ ones obtained is selected as the final solution of the matheuristic.

5. A branch-and-cut algorithm for the LC *K*-RPP

We have implemented a branch-and-cut algorithm for the LC *K*-RPP based on the formulation presented in Section 3. The initial LP is defined by all inequalities (2), (5), (7), and Eq. (4), while connectivity inequalities (3) and parity inequalities (8), which are exponential in number, are separated at each iteration of the cutting-plane algorithm and added to the LP. Let $\bar{x}^k,\ k = 1, \ldots, K$ be the fractional solution obtained at an iteration of the cutting-plane algorithm. We use the following separation algorithms:

separation of connectivity inequalities (3)

For each drone k, we compute the connected components of the graph induced by the edges $e \in E$ such that $\bar{x}^k_{ij} + \bar{x}^k_{ji} \geq 1 - \varepsilon$, where ε is a given parameter, and the depot, if necessary. For each connected component with node set S not including the depot, we select the edge $e = (i, j) \in E_R(S)$ with maximum value for $\bar{x}^k_{ij} + \bar{x}^k_{ji}$ and check the corresponding inequality $x^k(\delta(S)) \geq 2(x^k_{ij} + x^k_{ji})$ for violation. We start with $\varepsilon = 0$ and, while the algorithm fails in finding a violated inequality, we successively try $\varepsilon = 0.25$, 0.5, and 0.75.

Connectivity inequalities can be exactly separated with the following polynomial time algorithm. In the graph induced by the depot and the edges $e = (i, j) \in E$ such that $\bar{x}^k_{ij} + \bar{x}^k_{ji} > 0$, we compute, for each edge $e = (i, j) \in E_R$, the minimum cut separating edge e from the depot. If the weight of this cut is less than $2(\bar{x}^k_{ij} + \bar{x}^k_{ji})$, then the corresponding inequality (3) is violated.

Separation of parity inequalities (8)

Parity inequalities can also be separated in polynomial time by means of a procedure similar to the one proposed by Padberg and Rao (1982). However, since the procedure is time consuming, we do not use it in the cutting-plane algorithm. Instead, we use the following heuristic.

First, note that parity inequalities can be written as

$$\sum_{(i,j)\in\delta(S)\setminus F} (x^k_{ij} + x^k_{ji}) + \sum_{(i,j)\in F} (1 - x^k_{ij} - x^k_{ji}) \geq 1. \tag{9}$$

For each vehicle k, we compute the connected components of the graph induced by the edges $e = (i, j) \in E$ such that $\bar{x}^k_{ij} + \bar{x}^k_{ji} \geq 1 - \varepsilon$, where ε is a given parameter, and the depot, if necessary. For each connected component with node set S we check the edges in $\delta(S)$. For each $e = (i, j) \in \delta(S)$, if $\bar{x}^k_{ij} + \bar{x}^k_{ji} > 0,5$, we put e in F. If $|F|$ is odd, we are done. Otherwise, it is easy to determine the edge to be removed from or added to F, in such a way that the resulting set F minimizes the LHS of (9). If the LHS is less than 1, the inequality (9) is then violated. Otherwise there is no set $F \subseteq \delta(S)$ for which (9) is violated for the given set S. We start with $\varepsilon = 0$ and, while the algorithm fails in finding a violated inequality, we successively try $\varepsilon = 0.25$, 0.5, and 0.75.

6. Computational experiments

In this section, we present the instances used to analyze the behavior of the proposed matheuristic and branch-and-cut algorithms, as well as the computational study performed. The algorithms have been implemented in C++ and all the tests have been run on an Intel Core i7 at 3.4 gigahertz with 32 gigabytes RAM. The B&C uses CPLEX 12.6 MIP Solver with a single thread. CPLEX heuristic algorithms were turned off, and CPLEX's own cuts were activated in automatic mode. The optimality gap tolerance was set to zero and best bound strategy was selected. The branch-and-cut algorithm described in Corberán et al. (2007), used for obtaining the initial optimal giant tour and for optimizing the routes after the local search phase was also coded in C++ and uses CPLEX 12.6 MIP Solver too.

Table 1
Characteristics of the Campbell et al. (2018) Drone RPP instances.

Instance name	Original vertices	Original lines
DroneRPP56	22.3	21.3
DroneRPP66	27.0	23.3
DroneRPP58	34.0	29.6
DroneRPP68	36.6	35.0
DroneRPP77	38.6	41.6
DroneRPP510	41.6	42.0
DroneRPP610	50.0	46.6
DroneRPP79	50.3	53.6
DroneRPP88	56.3	51.0
DroneRPP710	58.3	56.6
DroneRPP89	60.3	56.3
DroneRPP99	66.3	65.3
DroneRPP810	68.6	67.0
DroneRPP910	78.6	74.6
DroneRPP1010	82.0	81.0

6.1. Instances

The two proposed procedures have been tested first on two sets of instances based on the ones proposed in Campbell et al. (2018) for the Drone RPP. The first set consists of 30 randomly generated instances, and the second set consists of 15 instances generated from the first set by replacing some required edges in order to reduce the number of odd-degree vertices, thus obtaining harder instances, called *even* instances. These all have between 22 and 83 original nodes and between 18 and 92 original lines (for details see Tables 2 and 3 in Campbell et al., 2018). Each row of Table 1 reflects three instances (two randomly generated and one even instance) and shows the average number of vertices and original lines of the three instances defined on a particular grid indicated by the digits at the end of the instance name. For example, the three instances for "DroneRPP710" have been generated on a grid with 7×10 points, and the original graphs (before any splitting) of these three instances have 58 vertices and 51 lines, 59 vertices and 65 lines, and 58 vertices and 54 lines, respectively.

In addition, we have generated 15 new larger instances as in Campbell et al. (2018), ten of which have been randomly generated, while the other five have been built trying to obtain few odd-degree vertices. The characteristics of these new instances are shown in Table 2. The first block, labeled "R", corresponds to the random instances and the "E" block to the even instances. Column 2 shows the name of the instances. The last digit in the name of each random instance indicates if this is the first or the second instance generated from the same grid. Columns 3 to 5 show the number of vertices and lines of the original instance and the number of non-required edges. Columns 6 and 7 give the number of vertices and required edges for the instance generated in the 1-splitting procedure of the matheuristic. The last five columns show the same data for the instances generated in the 3- and 7-splitting phases, plus the number of non-required edges in the 7-splitting case. The values in these last columns are average data, since the size of the 3-splitting (7-splitting) instances depends on the solution provided by the 1-splitting (3-splitting) procedure. Recall that the 3-splitting and 7-splitting are done only on a (small) subset of required lines and that the size of the resulting in-

Table 2
Characteristics of the new Drone RPP instances.

Type	Instance	Original vertices	Original lines	NR edges	1-splitting		3-splitting		7-splitting		
					Vertices	R edges	Vertices	R edges	Vertices	R edges	NR edges
	DroneRPP10111	88	72	3828	160	144	215.0	199.0	270.8	254.8	36549.8
	DroneRPP10112	93	98	4278	191	196	243.0	248.0	294.6	299.6	43289.2
	DroneRPP10121	95	80	4465	175	160	240.4	225.4	308.6	293.6	47500.0
	DroneRPP10122	102	110	5151	212	220	266.2	274.2	319.6	327.6	50960.8
R	DroneRPP11111	99	82	4851	181	164	250.8	233.8	322.4	305.4	51882.4
	DroneRPP11112	109	112	5886	221	224	280.8	283.8	340.4	343.4	57798.6
	DroneRPP11121	100	93	4950	193	186	258.6	251.6	325.0	318.0	52675.6
	DroneRPP11122	124	126	7626	250	252	320.0	322.0	391.6	393.6	76487.6
	DroneRPP12121	102	101	5151	203	202	262.2	261.2	322.4	321.4	51819.2
	DroneRPP12122	127	137	8001	264	274	337.2	347.2	408.4	418.4	83243.2
	DroneRPP1011	88	72	3828	160	144	214.8	198.8	270.2	254.2	36381.8
	DroneRPP1012	95	99	4465	194	198	244.0	248.0	295.4	299.4	43511.0
E	DroneRPP1111	99	106	4851	205	212	250.2	257.2	295.0	302.0	43392.8
	DroneRPP1112	100	109	4950	209	218	257.0	266.0	306.0	315.0	46698.0
	DroneRPP1212	102	114	5151	216	228	262.0	274.0	309.6	321.6	47775.0

Table 3
Computational results with the B&C on the LC K-RPP(0) instances.

	Number of			Gap0 (%)	Gap (%)	Nodes	Time UB	Time
	drones	inst	opt					
	2	20	19	3.47	0.04	821.5	77.0	258.8
	3	20	12	11.96	6.49	3133.5	1219.5	1651.9
R	4	20	4	21.52	16.83	7084.7	1919.9	2980.3
	5	20	2	27.68	24.50	5251.2	2441.2	3339.9
	6	20	3	28.74	25.40	2759.1	2744.0	3144.1
	2	20	18	1.88	0.22	399.6	624.4	1258.1
	3	20	1	10.16	9.38	989.1	2357.0	3429.1
R	4	20	0	22.80	22.25	547.6	2997.9	3600.0
	5	20	0	30.00	29.84	321.3	3028.5	3600.0
	6	20	0	36.06	35.95	182.2	3327.1	3600.0
	2	10	10	4.88	0.00	629.4	79.2	192.5
	3	10	5	16.95	10.17	5319.4	1858.6	2238.8
E	4	10	2	25.36	19.08	7471.6	1797.6	3183.9
	5	10	1	30.49	26.32	6151.0	2136.7	3470.6
	6	10	1	30.07	25.99	4034.6	2671.6	3291.6
	2	10	5	10.60	7.88	1695.4	1740.1	2663.1
	3	10	0	18.82	17.81	1119.1	2762.8	3600.0
E	4	10	0	27.69	27.15	620.8	3091.0	3600.0
	5	10	0	33.86	33.68	363.1	3314.9	3600.0
	6	10	0	36.77	36.52	216.3	3426.8	3600.0

stance would be much larger if this splitting were made in all the lines. For example, if 7-splitting were made on all the required edges, instance DroneRPP10122 would have 872 vertices, 880 required edges, and 379,756 non-required edges (independently of the number of drones used) instead of the 319.6 vertices, 327.6 required edges, and 50960.8 non-required edges on average.

The above are instances for the Drone RPP. In order to obtain instances for the LC K-DRPP we have to define the limit L for the length of the drone routes. To do so, several runs have been performed for each instance with different values for L in order to obtain solutions with a number of drones ranging from 2 to 6. Hence, we have 5 LC K-DRPP instances for each one of the 60 Drone RPP instances for a total of 300 instances. These instances are available at http://www.uv.es/corberan/instancias.htm.

6.2. Computational results

In this section we present the results obtained with the branch-and-cut and the matheuristic algorithms described in Sections 5 and 4, respectively, on the 300 LC K-DRPP instances presented before. All the results shown in what follows are given with respect to the deadheading cost of the solutions, which are the only ones that can be minimized.

6.2.1. Results obtained with the branch-and-cut algorithm

Table 3 summarizes the computational results obtained with the exact B&C algorithm for all the "random" (R) and "even" (E) LC K-RPP(0) instances with a time limit of 3600 seconds. The results for each type of instances are separated in two blocks according to the instance size (with those above being for up to 60 original vertices and those below being with 60 or more). Columns 2 to 4 contain the number of drones, the number of LC K-DRPP instances, and the number of the corresponding LC K-RPP(0) instances solved to optimality. Columns 5 and 6 show the average gaps in percentage between the cost of the optimal solution (if known) or that of the solution provided by the matheuristic before the 1-splitting phase and the lower bound at the end of the root node (Gap0) and the final lower bound (Gap), respectively. Column 7 reports the average number of nodes of the branching tree and the last two columns show the average computing time, in seconds, to reach the best feasible solution and the total time, respectively.

From Table 3 we can observe that the B&C is capable of solving almost all the instances with 2 drones (52 out of 60) in very short computing times and a good number of instances (18) with 3 drones. The average final gaps for the instances with 3 drones are quite good if we consider that in most cases they have been

Table 4
Sensitivity of the matheuristic to the parameter ℓ_{max}.

ℓ_{max}	Time	Deadheading cost	# Best
4	31.28	3103.74	8
6	34.04	3055.06	17
8	37.46	3039.11	19
10	42.49	3026.44	26

obtained comparing the lower bounds with upper bounds and not with optimal values. When the number of drones increases, the number of instances optimally solved decreases rapidly and becomes 0 in those instances with 60 vertices or more. Furthermore, the average gaps are far from good, which is a consequence of the great difficulty of this problem and a reason for the development of approximate algorithms for the LC *K*-DRPP, such as the one proposed here, capable of finding good solutions in reasonable times. Finally, as we expected, we can observe that the "even" instances present a greater difficulty than the "random" ones.

6.2.2. Results obtained with the matheuristic

We present here the computational results obtained with the proposed matheuristic on the set of LC *K*-DRPP instances described in Section 6.1.

First, in order to choose a value for the ℓ_{max} parameter (the maximum number of required edges used in the exchange moves), we tested the matheuristic on a representative subset of 30 LC *K*-DRPP instances with different sizes and number of drones with $\ell_{max} \in \{4, 6, 8, 10\}$ and $|\bar{S}_b| = 10$. Table 4 shows the obtained results. In this table, Column 2 shows the average computing time, in seconds, used by the algorithm to solve each instance with the different values of ℓ_{max}, and Column 3 shows the total deadheading cost on average. Column 4 reports the number of instances out of 30 for which the respective value of ℓ_{max} returns the best cost. In many cases, several values of ℓ_{max} get the same solution. Since the average time is not excessively larger, we chose $\ell_{max} = 10$. A similar test also suggested the value 10 for the parameter r_{max} (number of iterations without improvement of the destroy and repair algorithm).

To analyze the performance of the matheuristic, we have compared its results, using $|\bar{S}_b| = 10$, with those of the branch-and-cut algorithm on the LC *K*-RPP(0) instances. Note that the solutions obtained in the first phase of the matheuristic (no splitting has been made yet) are feasible solutions of the LC *K*-RPP(0) instances. Therefore, the lower bounds (or optimal values) obtained by the B&C are also lower bounds for the solutions obtained in the first phase of the matheuristic, and the upper bounds provided can also be compared to the matheuristic solutions. However, the solutions obtained in phases 2 and 3 of the matheuristic may be better than those obtained with the branch and cut, as in fact is the case in many instances.

Tables 5 and 6 show the results obtained for the "random" and "even" instances, respectively. Both tables present the same structure and, like Table 3, each one is separated into two groups according to the size of the instance. Columns 1 to 3 contain the number of drones used by the solution, the number of LC *K*-DRPP instances, and the number of the corresponding LC *K*-RPP(0) instances for which an optimal solution has been obtained with the branch and cut. Column 4 (GapLB) shows the percentage average gap between the cost of the solution provided by the matheuristic in phase 1 (0-splitting) and the lower bound (maybe the optimal value) given by the branch-and-cut algorithm. Column 5 (GapUB) provides the average percentage gap between the cost of the solution of the matheuristic (also before the 1-splitting phase) and that of the best solution found by the branch and cut in one hour of computing time. A negative value in this column means that, for that particular subset of instances, the solutions provided by the matheuristic have, on average, lower costs than the best feasible solutions found by the B&C. Although the solutions obtained with the matheuristic with the 1-, 3-, and 7-splitting procedures are not feasible solutions of the LC K-RPP(0) instances, they are also compared with the best feasible solutions found by the B&C in the time limit. The average gaps for these comparisons are shown in columns 6–8. The final column reports the average total computing time used by the matheuristic. The idea behind the last comparisons is to highlight the difference between the solutions for the original LC *K*-DRPP instances found by the matheuristic in less than three minutes on average and those obtained by the B&C in one hour of computing time.

We can observe in Table 5 that the solutions obtained by the matheuristic on the instances with 2 and 3 drones are very good, showing an average gap before the splitting procedures of less than 1.30% and 8.08%. With 4, 5 and 6 drones the gaps are high, but our guess is that these values are due more to the poor quality of the lower bound than to the quality of the solution produced by the algorithm. Note that only 9 optimal solutions are known for the random instances using 4 or more drones, and that all the average gaps with respect to the upper bounds are negative except for the instances with 2 drones, in which the majority of the upper bounds are optimal or near optimal. The 1-splitting, 3-splitting, and 7-splitting procedures improve the solutions in all the instances and the impact of each procedure on the result of the previous procedure is significant. For example, the values in the last row of Table 5 say that, for the 20 instances with 60 to 127 vertices and 6 drones, the costs of the solutions provided by the matheuristic in 92.04 seconds on average are, without splitting the required lines, 9.27% better than those obtained by the branch and cut in one hour, and 10.94%, 12.62%, and 13.49% better when the 1-splitting, 3-splitting, and 7-splitting procedures, respectively, are applied. On the other hand, note that the val-

Table 5
Results of the matheuristic for the "random" instances.

Number of			0-splitting		1-splitting	3-splitting	7-splitting	Total
drones	inst	opt	GapLB	GapUB	GapUB	GapUB	GapUB	time
2	20	19	0.92	0.88	−0.21	−1.10	−1.60	25.80
3	20	12	6.73	−1.01	−1.42	−3.00	−3.36	23.17
4	20	4	19.31	−2.76	−4.80	−6.34	−7.61	22.69
5	20	2	27.70	−5.61	−7.51	−9.24	−9.90	23.48
6	20	3	32.24	−10.10	−10.95	−13.06	−14.53	26.54
2	20	18	1.30	1.07	0.30	0.01	−0.25	131.45
3	20	1	8.08	−2.71	−4.36	−4.91	−5.16	113.32
4	20	0	16.82	−9.83	−11.35	−12.16	−12.45	98.79
5	20	0	29.46	−9.77	−11.92	−13.30	−13.86	97.16
6	20	0	42.76	−9.27	−10.94	−12.62	−13.49	92.04

Table 6
Results of the matheuristic for the "even" instances.

Number of			0-splitting		1-splitting	3-splitting	7-splitting	Total
drones	inst	opt	GapLB	GapUB	GapUB	GapUB	GapUB	time
2	10	10	2.10	2.10	−3.19	−4.81	−5.78	34.1
3	10	5	9.37	−2.94	−6.77	−7.64	−8.57	24.0
4	10	2	21.35	−3.92	−7.46	−9.46	−10.06	25.3
5	10	1	30.10	−6.57	−9.54	−11.04	−11.37	23.0
6	10	1	32.91	−6.49	−9.93	−11.48	−11.76	23.7
2	10	5	6.23	−2.62	−5.68	−6.52	−7.91	199.6
3	10	0	13.36	−7.24	−10.56	−12.45	−12.97	160.2
4	10	0	23.85	−10.63	−12.40	−15.26	−16.67	153.9
5	10	0	34.35	−11.76	−14.79	−16.32	−16.88	144.4
6	10	0	41.97	−10.76	−13.11	−14.46	−15.16	142.1

Table 7
Results for the instances with known optimal solutions.

	Number of			GapUB				Total
	drones	opt	opt M	0-splitting	1-splitting	3-splitting	7-splitting	time
R	2	37	25	0.86	−0.06	−0.53	−0.94	76.20
E		15	6	1.76	−2.99	−4.27	−5.20	55.83
R	3	13	7	1.35	1.07	−0.38	−0.82	22.55
E		5	4	0.86	−5.30	−6.42	−7.68	15.14
R	4	4	3	0.08	−0.23	−0.91	−4.94	14.86
E		2	0	2.55	−2.36	−2.44	−2.78	10.25
R	5	2	1	3.02	2.82	−1.48	−2.67	10.49
E		1	1	0.00	−2.72	−4.04	−4.30	7.84
R	6	3	3	0.00	−0.44	−1.33	−2.72	7.77
E		1	1	0.00	−6.40	−9.11	−9.38	11.45

ues in columns GapUB obtained for the larger instances are better than those obtained for the smaller ones. This could be explained by the poorer behavior of the branch and cut in these larger instances.

Similar comments can be made for the results shown in Table 6 for the "even" instances. The values in columns GapLB and the computing times for the larger instances are worse than for the "random" ones, thus supporting our proposition that, for the matheuristic algorithm, the "even" instances are harder than the "random" ones, as is also the case for the B&C algorithm.

Table 7 shows the results obtained by the matheuristic on the 83 instances for which an optimal solution for the corresponding LC K-RPP(0) instance is known. Columns 1 and 2 contain the instance type ("random" or "even") and the number of drones. Column 3 shows the number of LC K-RPP(0) instances with known optimal value and Column 4 reports the number among them for which phase 1 of the matheuristic provided the same solution. For example, among the 40 "random" instances with two drones, 37 have been solved to optimality by the branch and cut, while the matheuristic found 25 of these optima. Columns 5 to 8 (GapUB) show the average percentage gap between the cost of the solution of the corresponding phase of the matheuristic and that of the optimal solution found by the branch and cut. The last column reports the average computing time in seconds used by the matheuristic. From these results it can be observed that the gaps are very good for the instances with known optimal solution. In particular, note that the solutions provided by the 3- and 7-splitting phases are better than the optimal solutions obtained with the branch and cut on the LC K-RPP(0) instances.

Table 8 reports the effect of the local search and the exact route optimization procedures on the performance of the matheuristic in the 0-splitting phase. The first column gives the name associated with the three instances (two "random" and one "even" instance) generated in the same grid. Each of these three instances is solved with 2, 3, 4, 5 and 6 drones and, hence, each row in Table 8 contains the average data of 15 instances. The average number of vertices and original lines of each group of instances are reported in columns 2 and 3. Column 4 shows the average improvement (in percentage) obtained by the local search procedures (LS) in the 0-splitting phase with respect to the cost of the initial solutions, and Column 5 reports the average time in seconds. The last two columns show the same data corresponding to the route optimization procedure (RO) when applied to the solutions provided by the local search. It can be seen that the improvement obtained with the local search, 22.02% on average, is significant, and it does not depend on the size of the instance. However, the results seem to show a slightly increasing improvement for the route optimization procedure as the size of the instance increases.

Table 9 shows the contribution of the local search and route optimization procedures when the instances are grouped by number of drones. While the improvement obtained with the local search is similar for any number of drones, the impact of the route optimization procedure on the improvement of the solutions clearly decreases as the number of drones increases. This could be explained by the fact that, when the number of drones increases, the number of required edges in each route decreases, and there is less room for improvement.

On the other hand, it is interesting to point out that if the route optimization procedure is removed from the matheuristic, then the number of optima found decreases from 39 to 25 in the random instances, and from 12 to 8 in the even instances. Hence, the route optimization procedure plays an important role in our algorithm, both in terms of the improvement of the solutions and in the number of optima found, with a reasonable computing time.

To further analyze the contribution of the different local search components of the matheuristic, we have tested the algorithm on a subset of 30 instances of different sizes and number of drones. Each instance is run four times. Three times deactivating each of the three local search procedures and a fourth in which none is deactivated. Table 10 reports the results obtained. Column 2 shows

Table 8
Impact of local search and route optimization procedures (instances grouped by size).

Name	$\|V\|$	$\|E\|$	LS		RO	
			Imp (%)	Time	Imp (%)	Time
DroneRPP56	22.3	21.3	21.57	0.2	1.37	1.2
DroneRPP66	27.0	23.3	23.95	0.7	0.87	2.6
DroneRPP58	34.0	29.6	22.80	2.6	0.89	5.7
DroneRPP68	36.6	35.0	20.65	0.2	1.88	0.9
DroneRPP77	38.6	41.6	22.58	0.9	1.08	1.8
DroneRPP510	41.6	42.0	27.94	3.0	1.06	5.5
DroneRPP610	50.0	46.6	24.08	1.3	1.67	2.6
DroneRPP79	50.3	53.6	22.70	4.4	1.70	6.6
DroneRPP88	56.3	51.0	17.07	5.5	1.57	8.8
DroneRPP710	58.3	56.6	23.86	2.3	1.62	3.4
DroneRPP89	60.3	56.3	22.31	4.7	2.60	6.5
DroneRPP99	66.3	65.3	19.25	7.9	1.59	7.7
DroneRPP810	68.6	67.0	18.68	6.6	1.58	7.6
DroneRPP910	78.6	74.6	20.22	10.1	1.69	11.3
DroneRPP1010	82.0	81.0	22.96	13.3	2.71	12.9
DroneRPP1011	89.6	80.6	23.77	14.8	2.51	12.7
DroneRPP1012	97.3	96.3	25.59	28.4	1.85	19.3
DroneRPP1111	102.3	103.6	21.73	27.2	2.24	19.1
DroneRPP1112	108.0	109.3	16.89	29.3	2.38	19.7
DroneRPP1212	110.3	117.3	21.70	36.8	2.40	24.2

Table 9
Impact of local search and route optimization procedures (instances grouped by number of drones).

Number of		LS		RO	
drones	instances	Imp (%)	Time	Imp (%)	Time
2	60	22.32	9.2	3.40	9.6
3	60	22.71	9.7	1.99	7.9
4	60	22.52	10.6	1.59	8.4
5	60	21.82	10.4	1.01	9.1
6	60	20.71	10.2	0.82	10.0

Table 10
Analysis of the contribution of the different local search procedures.

Deactivated procedure	Time	Deadheading cost	# Best
None	51.44	2843.59	15
0 to ℓ exchange	32.81	2903.04	6
ℓ_1 to ℓ_2 exchange	36.65	2897.99	8
Destroy and repair	86.39	2849.48	9

the average time the algorithm takes to solve each instance deactivating the local search procedure indicated in Column 1, and Column 3 shows the deadheading cost on average for each case. Column 4 reports the number of instances out of 30 for which the respective solution mode returns the best deadheading cost of the four obtained. From the table we can see that the "0 to ℓ exchange" is the most effective procedure since when it is deactivated the algorithm shows its worst behavior, both in terms of the quality of the solutions (deadheading cost) and the number of best solutions found. Also the procedure "ℓ_1 to ℓ_2 exchange" is important to improve the solutions, while it seems that "destroy and repair" provides the least benefit in terms of deadheading cost, although disabling it makes the other methods take longer to find good solutions. Therefore, as confirmed by the results obtained when no local search method is deactivated, all procedures help to improve the solutions, which justifies their use in the proposed matheuristic.

Finally, for illustrative purpose, Fig. 6 shows the solution provided by the matheuristic for the even instance DroneRPP66 with six drones. Solid lines represent the traversal and service of the required lines, while dashed lines are associated with deadheading traversals. This solution has a deadheading cost of 3411, while

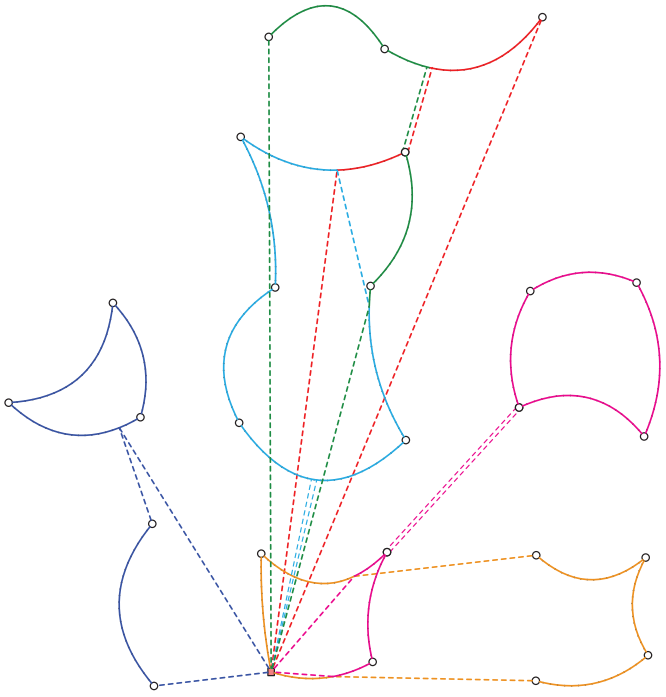


Fig. 6. Solution of an even DroneRPP66 instance with $L = 1300$ (6 drones).

the cost of the solution obtained for the same instance without splitting is 3751.85. Note that sometimes drones enter some required lines through intermediate points, and that the service of some lines is shared by two drones. The drone routes illustrated in Fig. 6 show some of the complexity of these problems and solutions, as well as the benefits the drones gain by being able to fly directly between two points. Observe that the green route in Fig. 6 includes parts of two required lines and all of two other required lines, which are in two separate components of the network. Note that these solutions also reflect the strategy of splitting the arcs in equi-distant segments, while a different set of intermediate points may lead to (likely small) improvements. In practice, user-driven selection of splitting points (e.g., based on local conditions) may be employed.

7. Conclusions

Drone arc routing problems (Drone ARPs) differ from classical Postman arc routing problems in that drones can fly directly between any two points without following the edges of the graph. This enables Drone ARPs to have better solutions than Postman ARPs. Here we have studied the Length Constrained K-Drones Rural Postman Problem, in which the limited capacity of the drones makes it impossible to service all the lines requiring service with a single drone. Therefore, we have to find a set of drone routes, each of limited length.

For this problem we have presented an ILP formulation and two solution methods, a branch-and-cut algorithm and a matheuristic. The B&C algorithm is based on the proposed formulation and on the strengthening of its linear relaxation through the separation of a family of valid inequalities that is exponential in number. The matheuristic consists of several features, including splitting edges to increase flexibility in servicing, local search to improve solutions, and an optimization procedure that exactly solves RPP instances associated with each single drone route. The algorithms have been tested on a large set of Drone RPP instances from the literature and on a new set of larger instances with up to 137 lines and with 2 to 6 drones.

Future research includes exploring some post-processing local improvements to selectively add more intermediate nodes to certain edges. Some interesting nodes to consider could be the interior points on a required edge that are closest to the endpoints of other required edges, or/and the points on a required edge that are closest to any point on other required edges.

We are also working on the theoretical study of the problem in order to find new families of valid inequalities that can help to improve the branch-and-cut algorithm. Another line of future research is the study of a more general problem that combines the visit of vertices and the traversal of certain lines in order to deliver goods and inspect areas, for example.

Acknowledgments

The work by Ángel Corberán, Isaac Plana, José M. Sanchis, and Paula Segura was supported by the Spanish Ministerio de Ciencia, Innovación y Universidades (MICIU) and Fondo Social Europeo (FSE) through project PGC2018-099428-B-I00.

References

Barahona, F., & Grötschel, M. (1986). On the cycle polytope of a binary matroid. *Journal of Combinatorial Theory, 40*, 40–62.

Bonnin-Pascuala, F., & Ortiz, A. (2019). On the use of robots and vision technologies for the inspection of vessels: A survey on recent advances. *Ocean Engineering, 190*, 106420.

Campbell, J. F., Corberán, A., Plana, I., & Sanchis, J. M. (2018). Drone arc routing problems. *Networks, 72*, 543–559.

Catapult, N. D. (2020). UASs (unmanned aerial system) to detect marine ingress near nuclear power stations. Accessed September 24, 2020. https://cp.catapult.org.uk/uass-unmanned-aerial-system-to-detect-marine-ingress-near-nuclear-power-stations-pathfinder/.

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

Chung, S. H., Sah, B., & Lee, J. (2020). Optimization for drone and drone-truck combined operations: A review of the state of the art and future directions. *Computers & Operations Research, 123*, 105004. https://doi.org/10.1016/j.cor.2020.105004.

Corberán, A., Eglese, R., Hasle, G., Plana, I., & Sanchis, J. M. (2020). Arc routing problems: A review of the past, present, and future. *Networks*. https://doi.org/10.1002/net.21965.

Corberán, A., & Laporte, G. (2014). Arc routing: Problems, methods, and applications, Philadelphia. In *MOS-SIAM series on optimization*. SIAM.

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

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms* (). MIT Press and McGraw-Hill.

Delair (2020). Delair long range surveillance drones help the French ministry of foreign affairs to better address counter terrorism in Niger. Accessed September 24, 2020. https://delair.aero/success-stories/delair-long-range-surveillance-drones-help-the-french-ministry-of-foreign-affairs-to-better-address-counter-terrorism-in-niger/.

Dille, M., & Singh, S. (2013). Efficient aerial coverage search in road networks. In *Proceedings of the AIAA guidance, navigation and control (GNC) conference*. https://doi.org/10.2514/6.2013-5094. August 2013.

United Nations ESCAP (2019). (Economic and social commission for asia and the pacific), working group on the trans-asian railway network. Inspection and monitoring of railway infrastructure using aerial drones. Note by the secretariat, ESCAP/TARN/WG/2019/4.

Franz, J. (2018). BNSF takes to the sky with drones to inspect montana main line; program may expand in 2019. November 14, 2018. Accessed September 27, 2020. https://trn.trains.com/news/news-wire/2018/11/14-bnsf-takes-to-the-sky-with-drones-to-inspect-montana-main-line-program-may-expand-in-2019.

Garrett-Glaser, B. (2019). Evergy inspects 150 miles of power lines with true BVLOS drone flights. November 13, 2019. Accessed September 24, 2020. https://www.aviationtoday.com/2019/11/13/evergy-inspects-150-miles-power-lines-true-bvlos-drone-flights/.

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

Hierholzer, C. (1873). Über die möglichkeit, einen linienzug ohne wiederholung und ohneunterbrechung zu umfahren. *Mathematische Annalen, 6*, 30–32.

Khoufi, I., Laouiti, A., & Adjih, C. (2019). A survey of recent extended variants of the traveling salesman and vehicle routing problems for unmanned aerial vehicles. *Drones, 66*(3), 3. https://doi.org/10.3390/drones3030066.

Knight, R. (2019). UAV inspection at the biggest oil rig in the world. December 2, 2019. Accessed September 24, 2020, https://www.microdrones.com/en/content/uav-inspection-at-the-biggest-oil-rig-in-the-world/.

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. https://doi.org/10.1016/j.cie.2018.05.039.

Liu, Y., Shi, J., Liu, Z., Huang, J., & Zhou, T. (2019). Two-layer routing for high-voltage powerline inspection by cooperated ground vehicle and drone. *Energies, 12*(7), 1385. https://doi.org/10.3390/en12071385.

Luo, H., Zhang, P., Wang, J., Wang, G., & Meng, F. (2019). Traffic patrolling routing problem with drones in an urban road system. *Sensors, 19*, 5164. https://doi.org/10.3390/s19235164.

Macrina, G., Di Puglia Pugliese, L., Guerriero, F., & Laporte, G. (2020). Drone-aided routing: A literature review. *Transportation Research Part C*. https://doi.org/10.1016/j.trc.2020.102762.

Mourão, M. C., & Pinto, L. S. (2017). An updated annotated bibliography on arc routing problems. *Networks, 70*, 144–194.

Networks (2019). Drone project begins flight trials on energy network assets. 16th May 2019. Accessed September 27, 2020- https://networks.online/gas/drone-project-begins-flight-trials-on-energy-network-assets/.

Oh, H., Kim, S., Tsourdos, A., & White, B. A. (2014). Coordinated road-network search route planning by a team of UAVs. *International Journal of Systems Science, 45*(5), 825–840.

Oh, H., Shin, H. S., Tsourdos, A., White, B. A., & Silson, P. (2011). Coordinated road--network search for multiple UAVs using dubins path. In F. Holzapfel, & S. Theil (Eds.), *Advances in aerospace guidance, navigation and control*. Berlin Heidelberg: Springer. 55–66

Otto, A., Agatz, N., Campbell, J., Golden, B., & Pesch, E. (2018). Optimization approaches for civil applications of unmanned aerial vehicles (UAVs) or aerial drones: A survey. *Networks, 72*, 411–458.

Padberg, M. W., & Rao, M. R. (1982). Odd minimum cut-sets and b-matchings. *Mathematics of Operations Research, 7*, 67–80.

Poikonen, S., & Campbell, J. F. (2020). Future directions in drone routing research. *Networks*. https://doi.org/10.1002/net.21982.

Press (2020). BVLOS powerline inspection over a city using VTOL UAVs. 16 July 2020. Accessed September 27, 2020. https://www.suasnews.com/2020/07/bvlos-powerline-inspection-over-a-city-using-vtol-uavs/.

Railway, B. (2019). It's a bird, it's a plane it's BNSF: How we lead the way in advanced drone operations. August 21, 2019, Accessed September 24, 2020, http://www.bnsf.com/news-media/railtalk/safety/bnsf-drone.html.

Seo, J., Dudue, L., & Wacker, J. (2018). Drone-enabled bridge inspection methodology and application. *Automation in Construction, 94*, 112–126.

Ulusoy, G. (1985). The fleet size and mix problem for capacitated arc routing. *European Journal of Operational Research, 22*, 329–337.

Zeisee, G. (2019). *BVLOS drones improve power line inspections amid increasing fire and storm risks for utilities*. May 17, 2019. Accessed September 24, 2020. https://www.utilitydive.com/news/bvlos-drones-improve-power-line-inspections-amid-increasing-fire-and-storm/554999/.