Universitat de València

Facultat de Ciències Matemàtiques

Departament d'Estadística i Investigació Operativa

Programa de Doctorat en Estadística i Optimització

Doctoral Thesis

Arc routing problems with drones

Paula Segura Martínez

Supervised by Ángel Corberán Salvador, Isaac Plana Andani and José María Sanchis Llopis

May 2023

How fortunate it is for me that my heart can feel the plain, naive delight of the man who puts on the table a cabbage that he has grown himself, and for whom it is not merely the vegetable, but all the good days, the fine morning when he planted it, the pleasant evenings when he watered it, taking his pleasure in its thriving growth, that he enjoys again in one comprehensive moment.

From *The Sorrows of Young Werther*,
Johann Wolfgang von Goethe

To my sister, my friend Míguel, and Ángel,
for being an essential support for me during these years

Contents

List of Figures

List of Tables

Introduction

The emerging technology of unmanned aerial vehicles, commonly known as *drones*, has provided new opportunities for practitioners in urban logistics in the last decade. Transportation plays a crucial role in society and economy, and a fundamental engine of economic development in recent times has been the investment on increasingly efficient transport systems. Drones present attractive advantages compared with standard ground vehicles, such as avoiding the congestion on road networks, eliminating the risk of personnel in difficult access operations or getting higher measurement accuracy in infrastructure inspection. Many commercial companies have recently shown interest in using drones for more cost–efficient and faster last–mile deliveries. Amazon announced at the end of 2013 that they would deliver packages directly to each doorstep through *Prime Air* using small drones 30 minutes after the "buy" button was hitted by the customers (CBSNews.com, 2013). As promised by the corporation, a later released version of its Prime Air delivery drone was a robust hybrid aircraft capable of vertical takeoff and landing that could fly up to 15 miles and deliver packages under five pounds to the customers in less than 30 minutes (Vincent and Gartenberg, 2019). Along with Amazon, other delivery services like UPS or Google have been testing the potential use of drones for parcel delivery (Mack, 2018). Since aerial drones are not restricted by local infrastructure, they can also be profitably used in rural distribution, surveillance and intralogistics (DHL, 2014), as well as environmental and geological 3D mapping for data collection (Sujit et al., 2012). The use of drones within all these scenarios faces multiple issues (and challenges) that can be addressed by routing problems, whose solution models aim to find the most efficient route (or routes) related to an explicit resource such as distance, time or energy.

This thesis focuses on the study of some extensions of arc routing problems in which such aerial vehicles, drones, are used to optimize a certain service. Given a graph representing a network, 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) of the graph, called *required* links, and satisfying certain conditions (see Corberán and Laporte, 2014; Mourão and Pinto, 2017; and Corberán et al., 2020). 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 a fleet of vehicles with limited capacity is available to jointly service all the required links. The main interest in studying this type of problems lies in the fact that they model a large number of real–life situations such as mail delivery, meter reading, garbage collection, network maintenance, and so on. ARPs can be formulated as combinatorial optimization problems, where the goal is to find, among a *countable* (but huge) set of feasible solutions, one that minimizes (or maximizes) a cost function, called *objective function.* Unfortunately, except for some of the simplest problems, it is not reasonable to expect to find algorithms capable of solving any instance of a problem in a number of operations that grows polynomially with the size of the instance.

The use of drones to perform the service in ARPs involves significant changes in the traditional way of modeling and solving these problems. In classical ARPs with ground vehicles, the streets requiring waste collection, roads from which snow must be removed, or pipelines to be inspected, for example, are represented by edges or arcs of a network that ignore the geometric shape of the modeled infrastructure (although not its cost or distance), since these vehicles have to traverse an arc or edge (a road) from one endpoint to the other one. Furthermore, vehicles in traditional ARPs cannot travel off the network. Instead, aerial drones have the capability to travel directly between any two points of the network, not necessarily between vertices of the graph. They may start the service of an edge at the most appropiate point along the edge according to the drone trajectory and the shape of each line to service. This consideration makes arc routing problems with drones continuous optimization problems with an infinite and *uncountable* number of feasible solutions.

Applications for drones in ARPs include inspection and monitoring of infrastructure and facilities that can be modeled as networks or collections of lines. Two relevant application areas are energy transmission systems and transportation. Jordan et al. (2018) provides examples of applications of drone inspection in a wide variety of areas, including power lines, railways, sewers, geographical features, buildings, bridges and wind turbines. In some areas such as power lines, the use of drones provides cost–effective, faster and safer inspections (Rauhakallio, 2020), and a new industry has developed to provide commercial power line inspection services. The academic research on power line inspection with drones is still limited. Liu et al. (2019) models the inspection of power lines, represented as straight line segments, by several drones that are launched from ground vehicles at a set of nodes on the road network, proposing constructive and improvement heuristics to design the routing planning of such a cooperative system. Other energy–related drone inspection topics receiving attention are wind turbine supervision, especially for expensive and difficult to access offshore wind farms (Durdevic et al., 2019); Shafiee et al., 2021), and monitoring of offshore oil and gas facilities and pipelines (Jones et al., 2019; Knight, 2019).

In transportation systems, drone arc routing examples arise in areas such as road traffic monitoring, railroad and transit track inspection, bridge and road inspection, and managing vegetation. The monitoring of urban traffic and road systems provides important applications for drone arc routes along infrastructure that naturally can be modeled with curved lines (see, for example, Li et al., 2018; Karaduman et al., 2019; and Luo et al., 2019). Bridges and other buildings associated with transportation systems also provide opportunities for drone arc routing to guarantee efficient inspections (Plotnikov et al., 2019; Outay et al., 2020). 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), in which case the area to be inspected by drones can be represented as a network of linear features (providing sensor coverage of the desired areas).

Throughout this thesis, we study three variants of arc routing problems with drones, which are modeled as combinatorial optimization problems and addressed with heuristic and exact mathematical approaches. The book is structured in seven chapters as follows. The first two are introductory chapters aimed at bringing the reader closer to essential notions and topics related to the content of the thesis. In Chapter 1, several basic definitions and theoretical results of graph theory, linear and integer programming, and combinatorial optimization are presented, and Chapter 2 provides an overview of the main routing problems studied in the academic literature.

The next three chapters deal with 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. In Chapter 3, the LC K–DRPP and also its discrete approximation, the Length–Constrained K–vehicles Rural Postman Problem (LC K–RPP), are defined. A formulation and some valid inequalities for the LC K–RPP are presented, as well as a branch–and–cut algorithm for its solution. A matheuristic algorithm for solving the LC K–DRPP is also developed, and extensive computational experiments to assess the performance of both algorithms are carried out.

Chapter 4 delves into the study of the LC K–RPP. We propose a new formulation for this discrete arc routing problem, whose binary variables are inspired by the work in Corberán et al. (2013) for the maximum benefit Chinese postman problem. Throughout this chapter, a polyhedral study of the set of solutions of a relaxed formulation is developed, proving that some inequalities from the formulation define facets of its associated polyhedron. Several families of valid inequalities are presented and some conditions under which such inequalities induce facets of the polyhedron are studied. In Chapter 5, a new algorithm for solving the LC K–DRPP is proposed. We present a new branch and cut for the LC K–RPP based on the formulation introduced in the previous chapter, which incorporates the separation of the valid inequalities proposed. This branch and cut is the main routine of an iterative algorithm that solves an LC K–RPP instance at each step in order to find good solutions for the original LC K–DRPP instance. The computational results obtained show the effectiveness of this new algorithm.

In Chapter 6, the Multi–Purpose K–Drones General Routing Problem (MP K–DGRP) is presented and studied. In this continuous optimization problem, a fleet of multi–purpose drones, aerial vehicles that can both make deliveries and conduct sensing activities, have to jointly visit a set of nodes to make deliveries and also map one or more continuous areas. We propose a matheuristic algorithm to solve the problem. Moreover, we define the Multi–Purpose K–vehicles General Routing Problem (MP K–GRP), a discrete optimization problem whose solution provides an upper bound of the optimal MP K–DGRP solution. For this problem, we present a mathematical formulation and several families of valid inequalities, and propose a branch–and–cut algorithm to solve it. Extensive computational experiments on two randomly generated sets of MP K–DGRP instances are carried out to test the performance of the proposed algorithms.

Chapter 7 presents the Load–Dependent drone General Routing Problem (LDdGRP). This problem is a variant of the classical GRP in which a drone has to traverse some required edges of a graph and also visit a set of nodes that require a delivery. Unlike basic GRP, where the total cost of the route is minimized and the cost of traversing each edge is constant, in the LDdGRP we aim to minimize the total flight duration, where we assume that the traversal time of an edge is proportional to the product of the distance travelled and the total weight (including cargo) of the drone. We propose a mathematical formulation for the problem and develop a theoretical study of the polyhedron associated with the LDdGRP solutions. Several families of valid inequalities are also proposed. We design a branch and cut for solving the LDdGRP that incorporates new separation procedures for all the proposed inequalities, and some computational experiments testing the efficiency of our exact approach are presented.

Finally, a section of conclusions is included as a recap of all the work developed in this thesis, and future lines of research are also outlined. Some of the original content appearing in this book has been published or submitted to international journals for publication. The content of Chapter 3 is based on the following published paper:

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

Part of the polyhedral study developed in Chapter 4 for the length constrained K–drones rural postman problem (the special case $K = 1$) can be found in

▷ Á. Corberán, I. Plana, J. M. Sanchis, and P. Segura (2021). Polyhedral study of a new formulation for the Rural Postman Problem. *Technical report*, http://www.uv.es/plani/reports.html

and a more compact version of the content of Chapters 4 and 5 is published in the following article:

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

Chapter 6 is based on the following paper already submitted to an international journal for publication:

▷ J. F. Campbell, Á. Corberán, I. Plana, J. M. Sanchis, and P. Segura (2023). The multi–purpose K drones general routing problem. *Under revision.*

The content of Chapter 7 will be submitted soon to a recognized journal in the area.

Chapter 1

Preliminary concepts

This chapter is aimed at introducing the reader to some basic concepts and results in the fields of graph theory, polyhedral theory, linear and integer programming, and combinatorial optimization, as well as presenting the mathematical terminology and notation used in the remaining chapters.

1.1 Graph theory

Many real–world problems, including vehicle routing problems, can be described and modeled by means of graph theory. We provide in this section some definitions and results of graph theory that will be used throughout the thesis. For further information in this field, we refer the reader to the excellent books by Harary (1969), Berge (1973), Christofides (1975), and Bondy and Murty (1976).

Let V be a non–empty finite set and let E be a finite family of unordered pairs (i, j) of (not necessarily distinct) elements in V. An *undirected graph* G is defined as a pair (V, E), where the elements of V are called *vertices* or *nodes* of G and the elements of E are called *edges* of G.

If the pair $e = (i, j)$ is an edge of G, then e is said to be *incident* with i and j, and vertices i and j are called the *ends* (or *endpoints*) of e, and are said to be *adjacents.* If an edge has identical ends, it is called a *loop.* If two edges are incident with the same pair of vertices, they are said to be *parallels.* A graph G is *simple* if it has no loops and no parallel edges. If G is simple and there is an edge (i, j) between each pair of distinct vertices i, j of G, it is said to be *complete.*

Given a vertex i of a graph G, we denote by $\delta_G(\{i\})$ (or simply $\delta(i)$, if there is no ambiguity) the set of edges of G that are incident with i. The *degree* $d_G(i)$ of vertex i is the number of edges of G incident with i, where each loop counts as two edges. A vertex of G is said to be *even* if it has even degree, and *odd* otherwise. A graph G is *even* if all its vertices are even.

Theorem 1.1.1. *An undirected graph G has an even number of vertices of odd degree.*

Let S_1 and S_2 be two disjoint subsets of V. We denote by $(S_1 : S_2)$ the set of those edges in G with one end in S_1 and the other end in S_2. For any non–empty set $S \subseteq V$, the set $\delta(S) = (S : V \setminus S)$ is called a *cutset* of G associated with S. In particular, each vextex $i \in V$ defines a cutset $\delta(i)$ of G. A cutset is said to be *even* if contains an even number of edges, and *odd* otherwise.

Theorem 1.1.2. *$\delta(S)$ is an odd cutset of G if, and only if, S and $V \setminus S$ have an odd number of vertices of odd degree.*

A graph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. Given a subset $V' \subseteq V$, we denote by $E(V')$ the set of those edges of G that have both ends in V'. The graph $G(V') = (V', E(V'))$ is called the subgraph of G *induced* by V'. Similarly, given a subset $E' \subseteq E$, we denote by $V(E')$ the set of vertices of G that are ends of an edge in E'. The graph $G(E') = (V(E'), E')$ is called the subgraph of G *induced* by E'.

Let us consider an undirected graph $G = (V, E)$ and two vertices i_0, $i_k \in V$. A *walk* from i_0 to i_k in G is a finite sequence $w = (i_0, e_1, i_1, e_2, \ldots, e_k, i_k)$ whose terms are alternately vertices and edges of G, such that the ends of each edge e_r are i_{r-1} and i_r, for $1 \leq r \leq k$. The integer k is the *length* of the walk. In a simple graph, a walk can be specified by the sequence $(i_0, i_1, i_2, \ldots, i_k)$ of its vertices, and if the vertices of this sequence are distinct, w is called a *simple* walk or *path*. A closed walk in G is called a *tour*.

Theorem 1.1.3. *Any tour in G traverses an even number of times (or zero) any cutset in G.*

Two vertices i and j of an undirected graph G are *connected* if there is a path from i to j in G. G is said to be *connected* if there is at least one path between each pair of vertices of G. A graph with a single vertex is connected by convention. A *connected component* of G is a connected subgraph of G induced by a set of vertices of G which is maximal with respect to the cardinality of such subset of vertices.

A *vertex cut* of an undirected connected graph G is a subset of nodes $V' \subseteq V$ such that the graph $G \setminus V'$ is disconnected. A vertex cut of k elements is called a *k–vertex cut.* A complete graph has no vertex cuts. An undirected graph G is said to be *k–connected* if k is the minimum number of vertices that must be removed for G to be disconnected. Similarly, an *edge cut* of G is a subset of edges $E' \subseteq E$ such that the graph $G \setminus E'$ is disconnected, and an edge cut of k elements is called a *k–edge cut.* The size of a minimum edge cut in a connected graph G gives its *edge connectivity.*

A *cycle* in G is a closed path. A *Hamiltonian path* of G is a path that contains every vertex of G, and a *Hamiltonian cycle* of G is a cycle that contains every vertex of G. A graph G is said to be *Hamiltonian* if it contains a Hamiltonian cycle. An *Eulerian tour* in graph G is a tour that traverses each edge of G exactly once. G is said to be *Eulerian* if it contains an Eulerian tour.

Theorem 1.1.4 (Euler). *An undirected graph G is Eulerian if, and only if, it is connected and even.*

Let V be a non–empty finite set and let A be a finite family of ordered pairs (i, j) of (not necessarily distinct) elements in V. A *directed graph G*, or *digraph*, is defined as a pair (V, A), where the elements of V are called *vertices* or *nodes* of G and the elements of A are called *arcs* of G.

Let V be a non–empty finite set, E a finite family of unordered pairs (i, j) of (not necessarily distinct) elements in V, and A a finite family of ordered pairs (i, j) of (not necessarily distinct) elements in V. A *mixed graph* G is defined as a tuple (V, E, A), where the elements of V are called *vertices* or *nodes* of G, the elements of E are called *edges* of G, and the elements of A are called *arcs* of G.

All the concepts already introduced for undirected graphs can be adapted to directed and mixed graphs.

1.2 Linear algebra and polyhedral theory

Let us denote by $\mathbb{R}^n$ the space of all the column vectors with n real components, and let $x_i \in \mathbb{R}^n$, with $i = 1\ldots, m$. A vector $y \in \mathbb{R}^n$ is said to be a *linear combination* of vectors x_i if, and only if, it can be expressed as $y = \sum_{i=i}^m \lambda_i x_i$, with $\lambda_i \in \mathbb{R}$. If $\sum_{i=i}^m \lambda_i = 1$ holds, y is said to be an *affine combination* of vectors x_i. A linear combination of vectors x_i satisfying $\lambda_i \geq 0$ is a *conic combination*, and an affine combination of vectors x_i in which $\lambda_i \geq 0$ is a *convex combination.*

Let us consider a set of vectors $X \subset \mathbb{R}^n$. The *linear* (resp. *affine*, *conic*, *convex*) *hull* of X is the set of all the linear (resp. affine, conic, convex) combinations of vectors in X, and it is denoted by $lin(X)$ (resp. $aff(X)$, $cone(X)$, $conv(X)$). X is said to be a *linear* (resp. *affine*, *conic*, *convex*) *subspace* of $\mathbb{R}^n$ if $X = lin(X)$ (resp. $X = aff(X)$, $X = cone(X)$, $X = conv(X)$).

Vectors $x_i \in \mathbb{R}^n$, $1 \leq i \leq m$, are said to be *linearly independent* if none of them can be written as a linear combination of the others, or equivalently, if $\sum_{i=1}^m \lambda_i x_i = 0$ implies that $\lambda_i = 0$ for each i. Otherwise, they are said to be linearly dependent. Similarly, vectors $x_i \in \mathbb{R}^n$, $1 \leq i \leq m$, are said to be *affinely independent* if none of them is an affine combination of the others, or equivalently, if $\sum_{i=1}^m \lambda_i x_i = 0$ with $\sum_{i=1}^m \lambda_i = 0$ implies that $\lambda_i = 0$ for each i. Otherwise, they are said to be affinely dependent.

Theorem 1.2.1. *Let us consider a set $X \subseteq \mathbb{R}^n$. The following statements are equivalent:*

i) X is affinely independent.

ii) For each $y \in X$, $\{x - y : x \in X, x \neq y\}$ is linearly independent.

iii) For each $y \in \mathbb{R}^n$, $\{x - y : x \in X\}$ is affinely independent.

The *range* of a set of vectors $X \subseteq \mathbb{R}^n$ is the maximum number of linearly independent vectors in X, and is denoted by $rg(X)$. The range of a matrix A, denoted by $rg(A)$, is the range of its column vectors, which coincides with the range of its row vectors.

If X is a linear subspace of $\mathbb{R}^n$, any finite subset B of linearly independent vectors in X such that $lin(B) = X$ is said to be a *base* of X. All the bases of a linear subspace of $\mathbb{R}^n$ have the same number of vectors, and this number is called the *dimension* of X. If X is an affine subspace of $\mathbb{R}^n$, there is a unique linear subspace $X' \subseteq \mathbb{R}^n$ such that $X' = \{x - y : x \in X\}$ for any $y \in X$, and the dimension of X' coincides with the dimension of X. If X is an arbitrary subset of $\mathbb{R}^n$, its dimension corresponds to the dimension of $\mathit{aff}(X)$, and is denoted by $\mathit{dim}(X)$.

Let $a \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$. A set $\{x \in \mathbb{R}^n : a^T x = \alpha\}$ is called a *hyperplane* in $\mathbb{R}^n$, and it divides $\mathbb{R}^n$ into two *half–spaces*, $\{x \in \mathbb{R}^n : a^T x \leq \alpha\}$ and $\{x \in \mathbb{R}^n : a^T x \geq \alpha\}$. A *polyhedron* in $\mathbb{R}^n$ is defined as the intersection of a finite number of half–spaces in $\mathbb{R}^n$ or, equivalently, as the set of solutions of a system of inequations of the form $Ax \leq b$, where A is an $m \times n$ matrix and $b \in \mathbb{R}^m$. If a polyhedron is bounded, it is called a *polytope.* A polyhedron $P \subseteq \mathbb{R}^n$ is said to be *full–dimensional* if $\mathit{dim}(P) = n$.

Let $a \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$. An inequality $a^T x \leq \alpha$ is a *valid inequality* for a polyhedron P if $P \subseteq \{x \in \mathbb{R}^n : a^T x \leq \alpha\}$. $F \subseteq \mathbb{R}^n$ is said to be a *face* of the polyhedron P if there exists a valid inequality $a^T x \leq \alpha$ for P such that $F = P \cap \{x \in \mathbb{R}^n : a^T x = \alpha\}$. It is said that face F is induced by the inequality $a^T x \leq \alpha$. A face F of a polyhedron P can be induced by different inequalities and, in this case, these inequalities are said to be equivalent with respect to P. Clearly, a polyhedron is a face of itself, and the other non–empty faces of P are called *proper faces.* A non–empty proper face of a polyhedron P that is maximal with respect to the inclusion of sets is called a *facet* of P.

Theorem 1.2.2. *Let $P \subseteq \mathbb{R}^n$ be a polyhedron and let F be a non–empty proper face of P induced by a valid inequality $a^T x \leq \alpha$. Then, F is a facet of P if, and only if, $\mathit{dim}(F) = \mathit{dim}(P) - 1$.*

A non–empty proper face of a polyhedron P that is minimal with respect to the inclusion of sets, that is, any face of P formed by a single point $F = \{v\}$, is called a *vertex* of P.

Theorem 1.2.3. *Let $P = \{x : Ax \leq b\}$ and $v \in P$. Then, v is a vertex of P if, and only if, v cannot be expressed as a convex combination of vectors in $P \setminus \{v\}$.*

A polyhedron P whose vertices have all their components integer is called an *integral polyhedron.*

1.3 Linear and integer programming

Linear programming dates back to the 1940s. In 1939, Leonid Kantorovich addressed the earliest linear programming problems that were used by the military during World War II to reduce army costs and increase battlefield efficiency. George B. Dantzig devised in 1947 the well–known *simplex method* to solve linear programming problems, and John von Neuman developed the theory of duality that same year.

A *linear programming problem* (LP) is the problem of minimizing (maximizing) a linear function

$$f(x) = c^T x, \; c^T \in \mathbb{R}^n,$$

called *objective function*, over a polyhedron $P = \{x \in \mathbb{R}^n : Ax \leq b\}$, where $x \in \mathbb{R}^n$ is a vector of *decision variables* and the linear inequalities $Ax \leq b$, with A an $m \times n$ matrix and $b \in \mathbb{R}^m$, are called *constraints* of the problem. Each $x \in P$ is a *feasible* solution of the LP, and we call *optimal* solution to any $x^* \in P$ such that

$$c^T x^* = \min\{c^T x : x \in P\}.$$

Theorem 1.3.1. *If P has at least one vertex and $\min\{c^T x : x \in P\}$ is finite, then there exists at least one vertex in P that is an optimal solution.*

The simplex algorithm (Dantzig, 1963) begins at a vertex of P and moves iteratively along the edges of the polytope until the vertex of an optimal solution is reached. Klee and Minty (1972) proved that this algorithm is not polynomial by finding a set of LP instances for which the simplex algorithm requires a number of operations that depends exponentially on the size of the instance. Khachian (1979) developed a polynomial algorithm called *ellipsoid method* to solve linear programming problems. The most important contribution of this method is the verification that LPs are problems in the so–called class $\mathcal{P}$. A few years later, Karmarkar (1984) proposed another polynomial method that has given rise to a whole family of algorithms, the so–called *interior point* methods, which move within the feasible region until an optimal solution is found. These methods rival the simplex in efficiency, especially on large instances. However, the simplex is still the most widely used algorithm in practice.

An *integer linear problem* (ILP) is the problem of minimizing (maximizing) a linear function

$$f(x) = c^T x,\ c^T \in \mathbb{R}^n,$$

with *integer decision variables* $x \in \mathbb{Z}^n$, over a polyhedron $P \in \mathbb{R}^n$. If the integrity constraint of the variables is omitted, an LP results that is called the *linear relaxation* of the ILP.

Integer linear problems are combinatorial optimization problems with a discrete set of feasible solutions, and although they may seem a simple subject at first sight, they can be extremely difficult to solve. For most combinatorial optimization problems, polynomial–time algorithms are not known, nor the existence of such algorithms is expected. They are known as NP–*hard*.

There is a variety of algorithms that can be used to deal with the ILPs exact solution. One of the most used is the *branch and bound.* After an optimal solution x^* of the LP relaxation is found, this method chooses some variable x_i that takes a fractional value x_i^* in the optimal solution of the LP, and generate one new subproblem with the additional constraint $x_i \leq \lfloor x_i^* \rfloor$ and a second one with the additional constraint $x_i \geq \lceil x_i^* \rceil$. Repeating this step on each new subproblem leads to the construction of a search tree whose nodes correspond to the linear relaxation of the original ILP with some additional constraints. The optimal values of the linear relaxations at the nodes of the tree are used as bounds to prune the tree, until an optimal integral solution is found.

Another procedure to solve ILPs exactly are the *cutting–plane* methods. The scheme of these algorithms is the following. We initially have an integer programming formulation of the problem $\min\{c^T x : x \in P, x \text{ is integral}\}$ for some polyhedron P, and some classes of inequalities that are valid for all the integral solutions. First, an optimal solution x^* of the LP relaxation $\min\{c^T x : x \in P\}$ is found with a linear programming algorithm. If x^* is integral, then it is also an optimal solution of the ILP. Otherwise, we look for valid inequalities of the known classes to find some that are violated by x^*. Then, these violated inequalities are added to the LP relaxation and a new optimal solution x^{**} is found. Again, if x^{**} is integral, we are done, and otherwise, we search for inequalities that are violated by x^{**}, add them to the current LP relaxation, and so on. One option when no longer violated inequalities are found in the root node is to proceed to branch. This generally more efficient method is called *branch and cut*, and combines the two optimization algorithms already mentioned, branch and bound and cutting–plane.

When ILPs contain a huge number of variables, they can be addressed with *branch–and–price* algorithms, which are a hybrid of branch and bound and *column generation* methods. At the beginning of this algorithm, a reformulation is used to define the *master problem*, and the LP relaxation of a restricted version of the master problem that considers only a subset of the columns is solved. Then, to check for optimality, a subproblem called *pricing problem* is solved to try to find columns with a negative reduced cost to be added to the restricted master problem, and the relaxation is re-optimized. When no profitable columns are found and the LP solution is not integral, the algorithm proceeds to branch and applies column generation at other nodes of the branch and bound tree until an optimal integral solution is found.

An alternative to exact methods for solving integer programming problems are the so–called *heuristic* algorithms. They provide a wide variety of non–exact procedures and strategies that, although cannot guarantee the best solution found to be an optimal one, can achieve a high–quality solution in reasonable computing time. These methods can be useful to obtain good bounds to prune the tree when ILPs are addressed with branching algorithms.

1.4 Polyhedral combinatorics

We summarize in this section the basis of the polyhedral approach for solving NP–hard combinatorial optimization problems.

Let us consider a finite set E, called *ground* set, with a cost c_e associated with each $e \in E$, and a finite (or countably infinite) family $\mathcal{F}$ of subsets of E, called feasible solutions. A *combinatorial optimization problem* (COP) with linear objective function is to find a set $F^* \in \mathcal{F}$ such that $c(F^*) = \sum_{e \in F^*} c_e x_e$ is minimum (or maximum), where x_e denotes the number of times that $e \in E$ is in F^*.

The polyhedral approach to the COP defined above starts with the definition of a polyhedron $P_{\mathcal{F}}$ whose vertices, and maybe other points in $P_{\mathcal{F}}$, follow a one–to–one correspondence with the feasible solutions in $\mathcal{F}$. For that, we define an incidence vector x^F of a feasible solution $F \in \mathcal{F}$ as

$$x^F = (x_e^F)_{e \in E} \in \mathbb{Z}^{|E|},$$

where x_e^F denotes the number of times $e \in E$ is in F, and then a convex set $P_{\mathcal{F}}$ is defined as

$$P_{\mathcal{F}} = conv\{x^F : F \in \mathcal{F}\}.$$

The set $P_{\mathcal{F}}$ is not a polyhedron in general (the set F can be infinite), but for most

COPs it can be shown that it is. We will therefore assume that $P_{\mathcal{F}}$ is a polyhedron. By construction, each feasible solution $F \in \mathcal{F}$ corresponds to a point in $P_{\mathcal{F}}$ and each vertex of $P_{\mathcal{F}}$ corresponds to a feasible solution $F \in \mathcal{F}$. Therefore, an optimal solution of the COP can be found by solving the LP

$$\min\{c^T x : x \in P_{\mathcal{F}}\}. \tag{1.1}$$

Thus, the COP has been naturally transformed into the LP (1.1), but in order to address problem (1.1) with the linear programming techniques, it is necessary to know the linear system that describes $P_{\mathcal{F}}$, or at least a significant part of it.

Even if a complete linear description of the polyhedron $P_{\mathcal{F}}$ is known, it is not possible to generate and store all the equations and inequalities that define the associated LP, since this linear system can be expected to be extremely large (in many problems, the number of inequalities grows exponentially with the number of variables in the problem). Instead of generating the entire linear system, what is done is to successively solve the following problem:

Separation Problem (or Facet Identification Problem). Given a polyhedron $P_{\mathcal{F}}$ and a point $\bar{x} \in \mathbb{R}^{|E|}$, either conclude that $\bar{x} \in P_{\mathcal{F}}$ or, if not, find a linear inequality $a^T\bar{x} \leq \alpha$ defining a facet of $P_{\mathcal{F}}$ that is violated by $\bar{x}$ (i.e., $a^T\bar{x} > \alpha$).

Using the facet identification problem as a subroutine in a cutting–plane algorithm, we could generate inequalities (which define facets of $P_{\mathcal{F}}$) that are violated as we need them to *cut off* optimal fractional solutions of the LPs. If a complete description of $P_{\mathcal{F}}$ is known and the separation problem can be solved, the cutting–plane ends at an optimal solution of the COP.

Theorem 1.4.1. *The optimization problem* (1.1) *is polynomially solvable if, and only if, the separation problem is polynomially solvable.*

Unfortunately, the complete description of the linear system that defines the polyhedron associated with an NP–hard problem is highly improbable (Papadimitrou, 1984) and, likewise, the separation problem of an NP–hard problem is also NP–hard. However, a partial knowledge of the $P_{\mathcal{F}}$ linear system and a "partial solution" of the separation problem may be enough to obtain important results if, when no violated inequalities are found, we start, for example, a branch–and–bound process. Thus, one important challenge in polyhedral combinatorics is to find a sufficiently large subsystem of the complete linear system that describes the polyhedron $P_{\mathcal{F}}$ in order to design efficient strategies to solve the problem.

Chapter 2

Routing problems overview

Routing problems are among the most studied problems in the last decades within the area of combinatorial optimization. This second introductory chapter focuses on bringing the reader closer to these problems, offering a summarized overview of their origin and classification, as well as some important definitions and relevant results from them.

A generic definition of the family of routing problems can be the following: given a vehicle (or a fleet of vehicles) and a set of transportation requests, to determine a route (or a set of routes) with minimum cost satisfying all the requests and meeting certain additional conditions. Many real world situations can be modeled as routing problems. For example, the distribution of goods to customers, the garbage collection over the streets, or the inspection and maintenance of highways or electrical infrastructures. These routing problems are modeled mathematically by representing the transport network on a graph, where each of its edges or arcs represents a connection (street, road, boundary, shoreline) of the real network and has an associated weigth representing the distance or cost of traversing it.

Classical routing problems can be naturally classified into two groups, namely node routing problems and arc routing problems, depending on where in the graph the service will be performed. In node routing problems, also called *vehicle routing problems* (VRPs), the service to be performed is located at the vertices of the graph. These problems would represent, for example, a real situation in which a traveler must visit a series of cities in such a way that the distance traveled is minimized. We will address this first group of problems in more depth in Section 2.1.

Consider now the situation in which a postman leaves his office, delivers the mail along the streets, and then returns to his office. Designing a route of minimum length for the postman is in this case equivalent to finding a tour on a graph where some edges (roads) have a certain demand and must be serviced. Those routing problems in which the solution must traverse a set of edges and/or arcs of the graph are called *arc routing problems* (ARPs). Section 2.2 is devoted to this second group of problems, providing a summary of the main ARPs defined in the literature, as well as some theoretical results and extensions of them. Arc and node routing problems can be further unified in the so–called *general routing problems*, as will be seen in Section 2.3, in which it is necessary to traverse a set of arcs, as well as visit a set of vertices of the graph.

Another way to classify routing problems could be based on the type of graph on which they are modeled. An *undirected* graph could represent a situation in which each road can be traversed at the same cost in both directions, a *directed* graph could model the case, for example, in which the roads can be traveled in only one direction, and a more flexible case where some streets can be traveled in only one direction and others in both could be represented in a *mixed* graph. Minieka (1979) introduced the *windy* postman problem as a generalization of the undirected, directed and mixed cases in which the cost of traversing an edge is not assumed to be the same on both directions. The name attributed to these problems arose from the idea of assuming one direction to be uphill (against the wind) and the other downhill (with the wind), and they could represent a frequent transport situation in which fares are different depending on the direction. Windy routing problems are modeled on an undirected graph where each edge (i, j) has two associated costs c_{ij}, c_{ji} (not necessarily equals) which represent the cost of traverse the edge from i to j or from j to i, respectively.

In many cases, a single postman (or vehicle) does not have enough capacity to meet all the demand, and the problem that arises is to find one route for each vehicle of a given set, so that the global demand is satisfied. Moreover, any of these routing problems may be complemented with additional conditions derived from the characteristics presented by the real situation they model, including time windows within which the service must be performed, multiple depots, precedence relationships of some services over others, split delivery, balancing of the routes, prohibited transitions from one arc or edge of the graph to another one, or some complex loading constraints, among others.

In recent years, the increasing advancement of aerial vehicle technologies has made drones very useful in some practical applications, such as transport of small packages, infrastructure inspection or border control. Either independently or in a cooperative environment with ground vehicles (trucks), these remote controlled devices offer new opportunities to improve logistics. We dedicate Section 2.4.1 to briefly comment on some of the new applications that appear when drones are used to solve routing problems, as well as some recent works and references in this new line of research. The main ideas and new considerations introduced in Campbell et al. (2018) to address arc routing problems with drones are summarized in some detail in Section 2.4.1, since they motivate part of the work described in the following chapters.

2.1 Node routing problems

In node routing problems, the transportation requests are concentrated in specific points of a road network, which is equivalent to assuming that a set of vertices of a graph needs to be visited by the solution of the problem at hand. There is a vast literature on node routing problems, much more extensive than for arc routing problems, motivated by their practical relevance and also their considerable difficulty. This section provides a review of the two most famous node routing problems, the traveling salesman problem and the vehicle routing problem, as well as some of their extensions and reference works in the area of vehicle routing. The interested reader can find more information on vehicle routing problems, solution methods and applications in the excellent book edited by Toth and Vigo (2014).

2.1.1 The traveling salesman problem

The *traveling salesman problem* (TSP) is almost certainly the most studied vehicle routing problem in the scientific literature within the area of combinatorial optimization.

Let us consider a complete weighted graph $G = (V, E)$, where the set of vertices V would represent a set of cities, the set of edges E would represent the roads between each pair of cities, and each weight w_e would be the cost or distance of road e. The TSP aims to find a Hamiltonian cycle with minimum weight on G, that is, the shortest route visiting each city exactly once and returning to the one of origin. Karp (1972) proved the TSP to be NP–hard.

The *graphical traveling salesman problem* (GTSP) was introduced in Cornuejols et al. (1985) and Fleischmann (1985, 1988) as a variant of the TSP in which the graph is not necessarily complete, and then the tour must visit each vertex of graph G *at least* once. The main advantages of the GTSP formulation compared to the classical formulation of the TSP are that the polyhedron of solutions is full–dimensional and fewer variables are necessary, in general, since we can work directly on the graph that models the road network, which is normally quite far of being a complete graph.

Basic variants of the TSP are the symmetric TSP, which is defined on a weighted *undirected* graph without loops, and the asymmetric TSP, which is defined on a *directed* graph. Most node routing problems were first introduced as variants of the TSP (that is, by considering the single vehicle case), but later extended to a multi–vehicle scenario. We will summarize some of these extensions in the next subsection directly for the case of several vehicles, the so called vehicle routing problem. Ilavarasi and Joseph (2014) provide a survey of variants of the TSP, and a detailed description of exact methods and computational studies for the TSP can be found in the book by Applegate et al. (2006).

2.1.2 The vehicle routing problem

The *vehicle routing problem* (VRP) is a generalization of the TSP introduced more than 60 years ago in Dantzig and Ramser (1959) as the *truck dispatching problem*, whose objective was the delivery of gasoline to service stations with a fleet of trucks. Dantzig and Ramser (1959) presented the first mathematical formulation for the VRP and also an algorithmic approach for its solution. A few years later, Clarke and Wright (1964) proposed an effective heuristic algorithm for solving the VRP, and such a combinatorial problem became very popular among researchers.

Given a set of customers and a fleet of vehicles located at a depot or warehouse, the VRP is to determine a set of routes with minimum cost to service all the customers with the given vehicle fleet. This generalization of the TSP involves deciding which vehicle visits which customer (that is, a partition of the set of customers) and in which order so that all vehicle routes can be *feasibly* executed. Therefore, it is also an NP–hard problem. The most studied version of the VRP is the capacitated VRP, which assumes that all vehicles have the same limited capacity to attend the customers demands. The wide variety and richness of methods proposed in the literature for the capacitated VRP are analized, in Laporte (1992), Toth and Vigo (2002), Cordeau et al. (2007), and

Baldacci et al. (2010, 2011), among others.

Many variants of the VRP have been studied in the scientific literature. One of them is the VRP with time windows (VRPTW), where the service at each customer must start within an associated time interval, which is called a *time window.* The VRPTW is also NP–hard, even if the number of vehicles is fixed (Savelsbergh, 1985). In the existing literature concerning VRPTWs, an unlimited number of vehicles are usually available to serve the customers, and while the objective for exact methods is to minimize the total traveled distance, many heuristics prioritize minimizing the number of vehicles used. We refer the reader to the surveys of Braysy and Gendreau (2005) and Baldacci et al. (2012) for additional information on heuristic and exact algorithms for the VRPTW.

Another important variant of the VRP is the family of *pickup and delivery* problems (PDPs) for the transportation of *commodities* or goods from different origins to different destinations. There are many categories of PDPs according to the considered type of demand and route structure: there may be one or several different types of goods involved, and these can be transported from a single or multiple depots to the customers, and also collected from the customers and transported back to one or several depots. PDPs have applications in, for example, repositioning of inventory, car sharing systems, collection of empty bottles and cans, or door–to–door transportation services. A general survey of PDPs in the literature is given in Parragh et al. (2008a, 2008b).

2.2 Arc routing problems

The origin of arc routing problems is attributed to the well–known Königsberg bridges problem. In the 18th century, the river Pregel divided the Prussian city of Königsberg into four parts which were connected by seven bridges a, b, c, d, e, f, and g (as can be seen in Figure 2.1), and it was popularly wondered if it was possible to arrange a route that would cross each bridge exactly once and then return to the starting point. This question was answered in the negative by Euler (1736), who argued that "when there are more than two areas in which an odd number of bridges leads, such journey is impossible". This problem can be posed as a routing problem in a graph with the different parts A, B, C, and D of the city represented by vertices and the bridges by edges, where we know that it is impossible to find an *Eulerian* circuit since the four vertices are odd. The study of Euler for this problem would lay the foundations for modern graph theory.

It was not until 1960 that the first publication related to an arc routing problem, the widely known Chinese postman problem, appeared. Guan (1962) stated the problem of designing the shortest distance route for a postman who had to walk through all the streets in a given neighborhood in order to deliver the mail, and proposed an (non-polynomial) algorithm for solving it. A few years later, this problem was shown to be solvable in polynomial time on some types of graph. In the earlier 1980s, a first overview of arc routing problems appears in the works of Assad et al. (1983) and Benavent et al. (1983), and Bodin and Golden (1981) offer a more detailed classification of these problems. Since then, arc routing field has evolved a lot within the area of combinatorial optimization, and many exact and heuristic algorithms that reach a high level of sophistication have been developed, allowing the solution of increasingly large instances

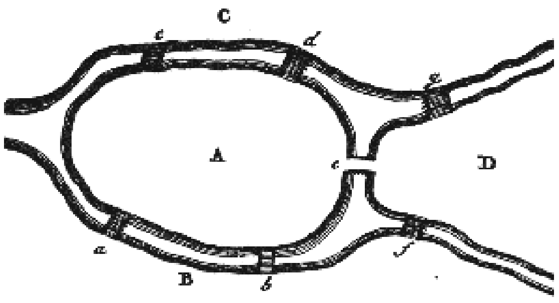


Figure 2.1: Euler's drawing of the river Pregel and the seven Königsberg bridges

in very reasonable computing times. The growing interest in the study of this type of problems has been motivated, in addition to its great theoretical appeal, by the large number of real–life situations they model, such as meter reading, newspaper delivery, snow plowing and salt spreading for winter maintenance of roads, waste management and collection along the streets, infrastructure inspection, and so on.

In the remaining of this section, we present a brief review of the most important arc routing problems studied in the scientific literature: the Chinese postman problem, the rural postman problem, and the capacitated arc routing problem. For each one of these problems, its definition on undirected graphs, which appears naturally depending on whether the demand is in all the edges of the graph, in only a subset of them, or whether a single vehicle can service the total demand or not, will be given. Their complexity on different types of graph and some of their most relevant extensions will also be mentioned. More information on definitions, methods and applications of arc routing problems can be found on the excellent book edited by Corberán and Laporte (2014). A recent annotated bibliography on arc routing problems is presented in Mourão and Pinto (2017), and a forecast of future lines of research in this area is provided in Corberán et al. (2021).

2.2.1 The Chinese postman problem

One of the most studied arc routing problems is the *Chinese postman problem* (CPP), which owes its name to the Chinese mathematician Meigu Guan, who introduced it in Guan (1962) more than 60 years ago. Given an undirected graph $G = (V, E)$ with a travel cost c_e associated with each edge $e \in E$, the CPP is to determine a least cost tour on G traversing each edge in E at least once. When the graph is connected and all its vertices are even, G is Eulerian and there exists a CPP solution which traverses each edge in E exactly once. Instead, if some vertices of G have odd degree, the problem is to identify a subset of edges to traverse twice, that is, to determine a least–cost augmentation of G that renders all degrees even.

In its undirected and directed version, the CPP is solvable in polynomial time and a complete description of its associated polyhedron is known (Edmonds and Johnson, 1973). The difficulty of the problem changes if it is defined on a mixed or windy graph. Papadimitriou (1976) shown the mixed CPP to be NP–hard. The windy postman problem, proposed in Guan (1984), includes as particular cases the previous ones and, therefore, it is NP–hard. Win (1989) proved the CPP to be solvable in polynomial time on a windy graph in certain special cases, such as when the graph is Eulerian.

Some variants of the CPP, many of them inspired from extensions of the TSP, appear in the scientific literature. Dror et al. (1987) introduced the hierarchical CPP, in which a precedence relation on arcs was considered, and developed an algorithm for solving it. The complexity of the algorithm of Dror et al. (1987) was reduced later in Ghiani and Improta (2000). Dror and Haouari (2000) proposed the generalized CPP, in which the set of edges is partitioned in several subsets and the solution has to contain at least one edge of each of such subsets. Aminu and Eglese (2006) introduced the CPP with time windows and proposed two exact constraint programming algorithms for its solution. A sophisticated branch and cut for the windy postman problem can be found in Corberán et al. (2012).

2.2.2 The rural postman problem

The *rural postman problem* (RPP) was first defined in Orloff (1974) as an extension of the CPP. Let us consider an undirected graph $G = (V, E)$ with a cost c_e associated with each edge $e \in E$, and let $E_R \subseteq E$ be a non–empty subset of edges of G, which will be called *required.* The goal of the RPP is to find a tour on G traversing each edge in E_R at least once with total minimum cost. Equivalently, the RPP aims at determining a least cost set of *deadheaded* (i.e., non–required) edges that, together with the set of required edges, generate an Eulerian graph.

The RPP on undirected and directed graphs was proven to be NP–hard in Lenstra and Rinnooy Kan (1976). However, when the set of required edges or arcs induces a connected graph, the RPP is reduced to the CPP and, therefore, it can be solved in polynomial time. As the RPP defined on mixed and windy graphs contain the undirected RPP as a special case, both are also NP–hard.

The first mathematical formulation for the RPP appears in Christofides et al. (1981). A constructive heuristic method for the RPP was proposed in Frederickson (1978), and several authors developed later alternative procedures (Hertz et al., 1999; Groves and van Vuuren, 2005; Ghiani et al., 2006). Blais and Laporte (2003) solved the directed RPP by transforming it into an asymmetric TSP. Since the general routing problem (GRP) generalizes the RPP, as will be seen in the next section, many results proposed for the first problem can be applied to the second one. Ávila et al. (2015) proposed a branch–and–cut algorithm for the directed general routing problem, and the branch and cut proposed in Corberán et al. (2007) for the windy general routing problem optimally solves the largest undirected RPP instances at present.

2.2.3 The capacitated arc routing problem

The *capacitated arc routing problem* (CARP) was introduced in Golden and Wong (1981) as a generalization of the RPP in which each edge of the graph has an associated demand, and a single vehicle does not have enough capacity to satisfy the total demand.

Let us consider an undirected and connected graph $G = (V, E)$ with a cost $c_e \geq 0$ and a demand $q_e \geq 0$ associated with each edge $e \in E$, a depot node, and a fleet of vehicles with the same capacity. The CARP is to find a set of routes for the vehicles with total minimum cost, all of them starting and finishing at the depot, and in such a way that customer demand is satisfied without exceeding the capacity of the vehicles.

Since the CARP contains the RPP as special case, it is also NP–hard. Belenguer and Benavent (1994) developed a branch and cut for the CARP based on a formulation with two indices, and a few years later developed a cutting–plane algorithm based on a one–index formulation (Belenguer and Benavent, 2003). The first column generation approach for the CARP appears in Gómez–Cabrero et al. (2005). Baldacci and Maniezzo (2006) transformed the CARP into a capacitated VRP and solved it with an exact method.

The CARP has also been defined for directed and mixed graphs. Belenguer et al. (2006) presented a cutting–plane algorithm which embeds the one–index formulation for the undirected CARP adapted for the mixed case, and Gouveia et al (2010) proposed a formulation for the mixed CARP that considers flow variables. Several constructive heuristics and metaheuristics algorithms have also been proposed for the CARP in the literature, many of them based on the classic path–scanning and augment–merge algorithm (Golden et al., 1983), or in the Ulusoy method (Ulusoy, 1985).

2.3 General routing problems

The most general case among routing problems is the one in which the service demand is found both in the links and in the nodes of the graph, namely the *general routing problem* (GRP).

The GRP was introduced in Orloff (1974), and its undirected version is defined as follows. Let us consider an undirected and connected graph $G = (V, E)$ with a cost c_e associated with each edge $e \in E$, let $E_R \subseteq E$ be a subset of *required* edges of G and let $V_R \subseteq V$ be a subset of *required* nodes of G. The GRP is to find the minimum cost tour on G which traverses each edge $e \in E_R$ and visits each node $v \in V_R$ at least once. Note that the GRP includes as special cases most of the problems already defined above. When the set of edges E_R is empty and $V_R = V$, the GRP becomes the GTSP. If $V_R = \emptyset$, it is reduced to the RPP, and if $E_R = E$, then the CPP arises.

The GRP was also proven to be NP–hard in Lenstra and Rinnooy Kan (1976). Since then, the problem has been studied in undirected, directed, mixed, and windy graphs (Corberán and Sanchis, 1994, 1998; Letchford, 1997, 1999; Corberán et al., 2001, 2003, 2005, 2007). Some variants of the GRP studied in the literature are the capacitated GRP in mixed graphs (Pandi and Muralidharan, 1995; Bosco et al., 2013), also known as the *node, edge and arc routing problem* (Bach et al., 2013), or the undirected capacitated GRP with profits (Archetti et al., 2017).

2.4 Routing problems with drones

So far this century, research dedicated to the use of aerial drones in transportation has grown in importance along with continuous technological advances in the drone industry. Drones are increasingly used by large companies to perform tasks such as traffic monitoring, last–mile delivery, remote sensing or aerial inspection, and new real world situations require new models that better reflect their characteristics.

Several extensions of the node routing problems described above that consider the use of drones have been proposed in the scientific literature. The first version of the TSP with drones, or TSP–D, was introduced in Murray and Chu (2015) by combining the service of some customers (those requiring heavier weight packages) by a truck and of the others by a drone (without cooperation). Agatz et al. (2018) proposed two heuristic approaches for a TSP–D that considers synchronization between both types of vehicle involved (truck and drone). These two works have served as a basis for a large number of scientific articles addressing TSP–D variants in recent years.

Wang et al. (2017) presented a generalization of the TSP–D, the VRP–D, that considers a fleet of trucks equipped with drones that can be dispatched from and picked up at any node (location) of the network. The authors conducted the analysis of several worst–case scenarios, and later Wang and Sheu (2019) presented an arc–based model and proposed a branch–and–price algorithm for the problem. Di Puglia Pugliese and Guerriero (2017) extended the VRP–D by considering time windows for each customer, and Ulmer and Thomas (2018) presented a dynamic variant of the VRP–D. Other extension of the classical VRP is the *drones delivery problem* (DDP), in which the fleet is composed only by drones, and several particular technical aspects of the drones, such as battery capacity or energy consumption, are taken into account (Dorling et al., 2017; Coelho et al., 2017; Troudi et al., 2019).

For a more extensive insight into recent work on node routing problems with drones, we suggest the reader the recent surveys of Macrina et al. (2020) and Rojas Viloria et al. (2021).

2.4.1 Drone arc routing problems

Unlike the growing scientific interest in addressing different versions of drone VRPs, references to arc routing problems with drones in the academic literature are still scarce. In transportation systems, drone arc routing examples arise in areas such as road traffic monitoring (Li et al., 2018; Luo et al., 2019), railway inspection (Plotnikov et al., 2019; Wishart et al., 2020), and roadway surface inspection (Outay et al., 2020), in which infrastructure is naturally modeled with curved line features.

Campbell et al. (2018) presented the first approach to drone arc routing problems (Drone ARPs) and studied their relation with classical ARPs. As pointed out in Campbell et al. (2018), Drone ARPs extend classical ARPs to allow vehicles (drones) to travel directly between any two points on the given network. In traditional ARPs, the vehicles travel through the edges of a network, and each edge has to be completely traversed (from one of its endpoints to the other one), or not traversed. However, while ground vehicles are limited to following the local infrastructure (roads), drones have the flexibility to travel off the network as well. These aerial vehicles may service only part of an edge and then travel in a straight line to any point of another edge, without following the links of the network. As a consequence, Drone ARPs are *continuous optimization problems* with an infinite number of feasible solutions.

Since drones may start and end the service of each edge of the network at any point of it (even in the middle), another relevant difference of Drone ARPs with respect to classical ARPs is that now the geometric *shape* of the lines must be taken into account in the construction of the routes.

With all the above assumptions, Campbell et al. (2018) defined the rural postman problem with drones, or Drone RPP, as follows:

> (**Drone RPP**) *Given a set of lines, each one with an associated service cost, and a point called the depot, and assuming that the cost of deadheading between any two points is the Euclidean distance, find the minimum cost tour starting and ending at the depot that services all the given lines.*

The mathematical approach developed by the authors for its solution was based on approximating each curve in the plane by a polygonal chain with a *finite* number of segments, and solving the problem as a *discrete optimization problem* (a postman RPP), where vehicles are allowed to enter and leave each curved line only at the points of the polygonal chain. Once discretized, the set of non–required edges in Drone ARPs forms a *complete* graph, and the deadheading cost between any pair of points is given by the Euclidean distance.

The work presented in the next chapters of this thesis extends the drone RPP defined in Campbell et al. (2018) to assume that the autonomy (flight range) of the drones is limited and a fleet of (homogeneous) drones is needed to perform all the service. Other related problem in the literature can be found in Amorosi et al. (2021), where the authors study the coordination problem that arises between a mothership vehicle and a drone that can be launched from it to (partially) perform inspection activities over a two–dimensional space, a connected piecewise linear polygonal chain, or a general graph.

Chapter 3

The length constrained *K*–drones rural postman problem

This chapter addresses the first problem studied in this thesis, the length constrained K–drones rural postman problem (LC K–DRPP). This problem was introduced in Campbell et al. (2018) as an extension of the Drone RPP in which the limited flight range of the drones implies that a single vehicle cannot perform all the required service, and is defined as follows:

> (**LC *K*–DRPP**) *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.*

Unlike ground vehicles in traditional 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: the LC K–DRPP is a *continuous* optimization problem, with an uncountable number of feasible solutions.

As carried out in Campbell et al. (2018), one way to deal with this problem is to digitize each specific instance by defining each curved line by a (usually) large set of points, each one with its corresponding coordinates. In other words, each line can be approximated by a polygonal chain in such a way that 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 combinatorial optimization arc routing problem, 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 (segment) $e \in E_R$, $c_e^s \geq 0$, 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 $e' \in E_{NR}$. It is assumed that $c_e^s \geq c_{e'}$. 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 when 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 1553203 non–required edges. 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). We will call LC K–RPP(0) to these instances. 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 traditional ARPs the graph often corresponds to a sparse network.

The contributions of this chapter are arranged as follows. In Section 3.1, we propose a formulation with binary variables and some families of valid inequalities for the discrete LC K–RPP, and a branch–and–cut algorithm (B&C) based on the proposed formulation is presented in Section 3.2. Section 3.3 describes a matheuristic algorithm for the LC K–DRPP resolution. We report in Section 3.4 the extensive computational experiments carried out to assess the performance of both algorithms on two sets of instances based on the ones proposed in Campbell et al. (2018) for the Drone RPP and also on 15 new larger instances generated in this work.

3.1 A formulation for the length–constrained K–vehicles rural postman problem

In this section, we present a formulation for the LC K–RPP on the graph $G = (V, E)$ described above. Since (V, E_{NR}) is a complete graph, G is an undirected multigraph with a non–required edge e' parallel to each required edge e, with different costs $c_{e'}$ and c^s_e. Given that G is undirected, 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.

The LC K–RPP can be formulated 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:

$$\text{Minimize} \sum_{k=1}^{K} \sum_{e=(i,j)\in E_{NR}} c_e(x^k_{ij} + x^k_{ji}) + \sum_{k=1}^{K} \sum_{e=(i,j)\in E_R} c^s_e(x^k_{ij} + x^k_{ji}) \tag{3.1}$$

s.t.:

$$\sum_{(i,j)\in\delta(i)} (x^k_{ij} - x^k_{ji}) = 0, \quad \forall i \in V,\ \forall k \in \{1, \ldots, K\} \tag{3.2}$$

$$x^k(\delta(S)) \geq 2(x^k_{ij} + x^k_{ji}), \quad \forall S \subset V \setminus \{1\},\ \forall (i,j) \in E_R(S),\ \forall k \tag{3.3}$$

$$\sum_{k=1}^{K}(x^k_{ij} + x^k_{ji}) = 1, \quad \forall (i,j) \in E_R \tag{3.4}$$

$$\sum_{e=(i,j)\in E_{NR}} c_e(x^k_{ij} + x^k_{ji}) + \sum_{e=(i,j)\in E_R} c^s_e(x^k_{ij} + x^k_{ji}) \leq L, \quad \forall k \in \{1, \ldots, K\} \tag{3.5}$$

$$x^k_{ij}, x^k_{ji} \in \{0, 1\}, \quad \forall (i,j) \in E,\ \forall k \in \{1, \ldots, K\} \tag{3.6}$$

The objective function (3.1) 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 (3.4), 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 (3.2) force each drone k to exit a vertex i as many times as it enters it. Connectivity inequalities (3.3) ensure each single route is connected and connected to the depot, while constraints (3.5) 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 equations (3.4). Constraints (3.6) are the binary conditions for the variables.

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

$$x^k_{ij} + x^k_{(ij)'} \leq 1. \tag{3.7}$$

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

$$x^k(\delta(S)\setminus F) \geq x^k(F) - |F| + 1, \quad \forall S \subset V,\ \forall F \subseteq \delta(S) \text{ with } |F| \text{ odd},\ \forall k \in \{1,\dots,K\} \tag{3.8}$$

These inequalities are based on the fact that all vehicles have to traverse any edge cut–set 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 (3.8) are referred to as cocircuit inequalities by Barahona and Grötschel (1986) and were proposed for the RPP by Ghiani and Laporte (2000).

3.2 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 3.1. The initial LP is defined by all inequalities (3.2), (3.5), (3.7), and equations (3.4), while connectivity inequalities (3.3) and parity inequalities (3.8), which are exponential in number, are separated at each iteration of the cutting–plane algorithm and added to the LP. Let $\bar{x}^k$, for $k = 1,\dots,K$, be the fractional solution obtained at an iteration of the cutting–plane algorithm. We use the following separation algorithms.

Separation of connectivity inequalities

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 $\bar{x}^k(\delta(S)) \geq 2(\bar{x}^k_{ij} + \bar{x}^k_{ji})$ 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 $e \in E_R$, 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 (3.3) is violated.

Separation of parity inequalities

Parity inequalities can also be separated in polynomial time by means of a procedure similar to the one proposed in 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 (3.8) can be written as

$$\sum_{(i,j)\in\delta(S)\setminus F}(x_{ij}^k + x_{ji}^k) + \sum_{(i,j)\in F}(1 - x_{ij}^k - x_{ji}^k) \geq 1. \tag{3.9}$$

For each vehicle k, we compute the connected components of the graph induced by the edges $e = (i,j) \in E$ such that $\bar{x}_{ij}^k + \bar{x}_{ji}^k \geq 1 - \varepsilon$, 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 $\bar{x}_{ij}^k + \bar{x}_{ji}^k > 0,5$, 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 (3.9). If the LHS is less than 1, the inequality (3.9) is then violated. Otherwise there is no set $F \subseteq \delta(S)$ for which (3.9) 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.

3.3 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:

i) 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. First, the algorithm 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 3.3.1. Three local search procedures are then applied to each of these solutions to improve the routes (described in Section 3.3.2), and finally each of the single routes of the resulting solutions are optimized (described in Section 3.3.3).

ii) 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 3.3.4.

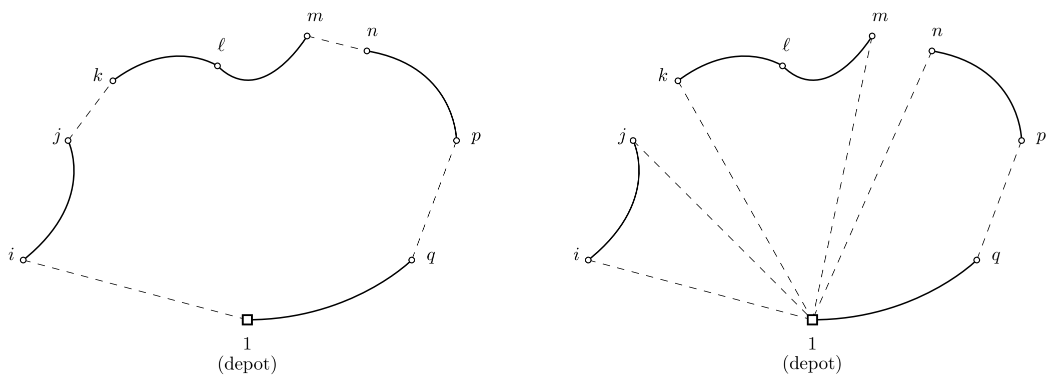
iii) In phase 3, the most "promising" segments of each solution obtained in phase 2 are split again by adding one new intermediate vertex to them, while some unused intermediate vertices are removed. This procedure is called "3–splitting" because some of the original lines are approximated by a polygonal 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 as final solution of the algorithm. This is described in Section 3.3.5.

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 $\{(i,j),(k,l),\ldots,(u,v)\}$ 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 needed non–required edges. Throughout this section we will denote by T_i the i–th route in a given solution S, with $i \in \{1,2,\ldots,K\}$.

3.3.1 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 in Corberán et al. (2007). This giant tour (see Figure 3.1a) is then partitioned into K routes of length no greater than L (see Figure 3.1b) by means of the procedure proposed by Ulusoy (1985) for the 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 Figure 3.2). 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}).



(a) Giant tour T_G on G (b) Resulting K routes

Figure 3.1: Splitting an optimal giant tour T_G into K feasible routes

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 Figure 3.2).

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 Figure 3.2, 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.

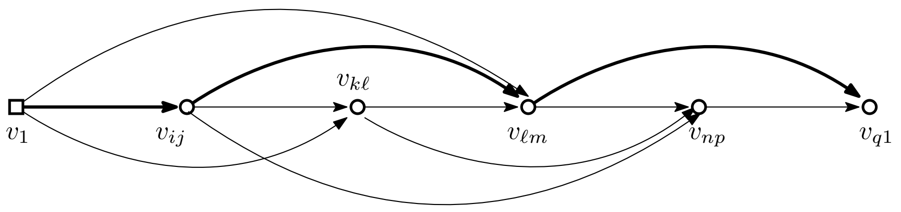


Figure 3.2: Auxiliary graph G^* generated from $T_G = \{(i, j), (k, \ell), (\ell, m), (n, p), (q, 1)\}$

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 (Cormen et al., 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 Figure 3.2, represented in bold lines, corresponds to the K feasible routes depicted in Figure 3.1b. 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 $|E_R|$ times, starting each time with a different required edge (Hierholzer, 1873). We thus obtain a set $\bar{S}$ of different initial LC K–RPP(0) solutions, with $|\bar{S}| \leq |E_R|$.

3.3.2 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.

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.

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.

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.

3.3.3 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 (plus the depot). 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.

3.3.4 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 (equidistant 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, Figure 3.3 shows an example of the behavior of a solution before (Figure 3.3a) and after (Figure 3.3b) 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.

3.3.5 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

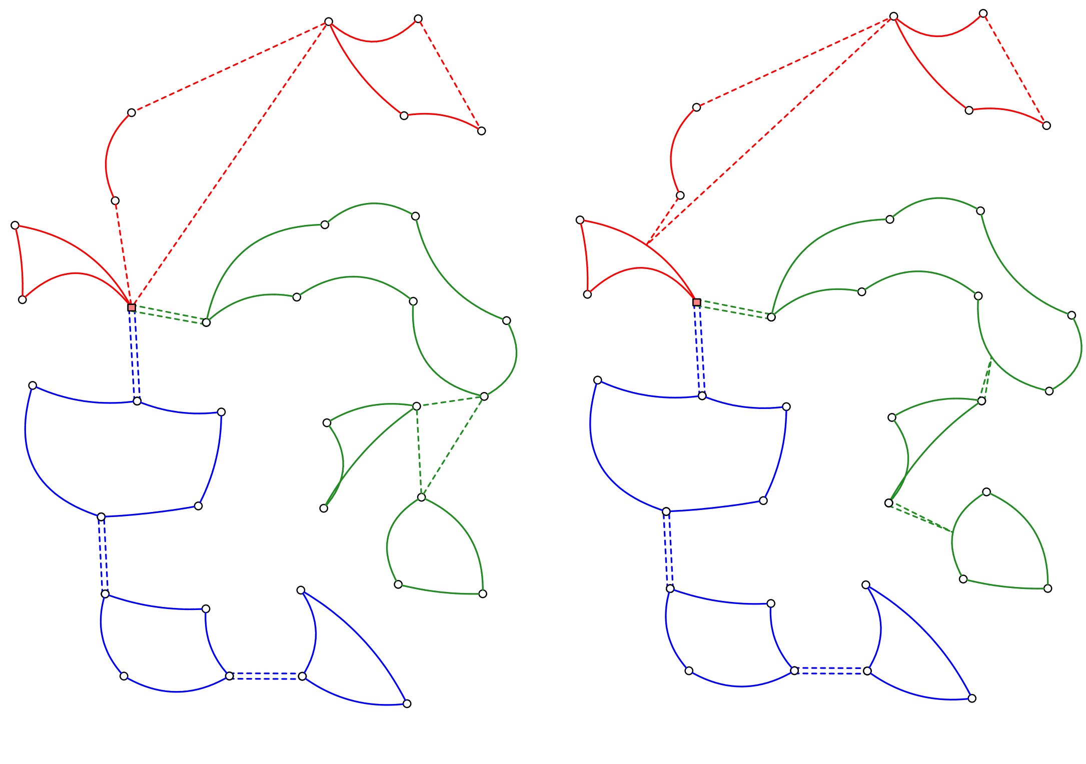
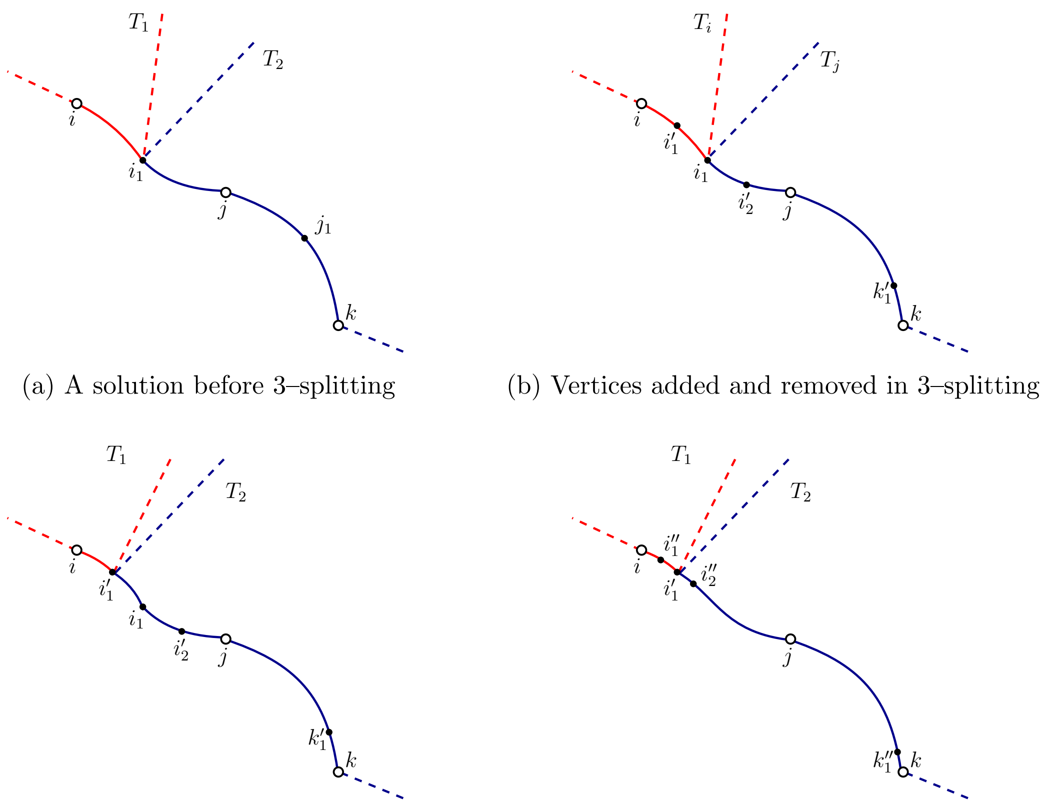

(a) Deadheading cost: 1808.24 (b) Deadheading cost: 1697.51

Figure 3.3: Two solutions of a DroneRPP68 instance before and after the 1–splitting procedure

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 Figure 3.4b).

Figures 3.4a and 3.4b 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 (Figure 3.4b) 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



(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

Figure 3.4: Illustration of 3– and 7–splitting

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 Figure 3.4c, 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. Figures 3.4c and 3.4d show how the 7–splitting works. Figure 3.4d 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 applied and the best solution among the $|\bar{S}_b|$ ones obtained is selected as the final solution of the matheuristic.

3.4 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

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

Table 3.1: Characteristics of the Campbell et al. (2018) Drone RPP instances

run on an Intel Core i7 at 3.4 GHz with 32 GB 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.

3.4.1 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 (complete details can be seen in Tables 2 and 3 in Campbell et al., 2018). Each row of Table 3.1 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

		Original	Original		**1–splitting**		**3–splitting**		**7–splitting**		
Type	Instance	vertices	lines	NR edges	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 3.2: Characteristics of the new Drone RPP instances

to obtain few odd–degree vertices. The characteristics of these new instances are shown in Table 3.2. 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 instance 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 379756 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/plani/instancias.htm.

3.4.2 Computational results

In this section, we present the results obtained with the matheuristic and the branch–and–cut algorithms described in Sections 3.3 and 3.2, 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.

Results obtained with the branch–and–cut algorithm

Table 3.3 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 by a horizontal line, according to the instance size (with those above the line being for up to 60 original vertices and those below the line 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

	K	♯ inst	♯ opt	Gap0 (%)	Gap (%)	Nodes	Time UB	Time
	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

Table 3.3: Computational results with the B&C on the LC K–RPP(0) instances

the last two columns show the average computing time, in seconds, to reach the best feasible solution and the total time, respectively.

From Table 3.3 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 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.

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 3.4.1.

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 3.4 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} reported in Column 1, 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).

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

Table 3.4: Sensitivity of the matheuristic to the parameter ℓ_{max}

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 3.5 and 3.6 show the results obtained for the "random" and "even" instances, respectively. Both tables present the same structure and, like Table 3.3, each one is separated into two groups by a horizontal line, 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 to 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

			0–splitting		**1–splitting**	**3–splitting**	**7–splitting**	
K	♯ inst	♯ opt	GapLB	GapUB	GapUB	GapUB	GapUB	Time
2	20	19	0.92	0.88	−0.21	−1.10	−1.60	25.8
3	20	12	6.73	−1.01	−1.42	−3.00	−3.36	23.2
4	20	4	19.31	−2.76	−4.80	−6.34	−7.61	22.7
5	20	2	27.70	−5.61	−7.51	−9.24	−9.90	23.5
6	20	3	32.24	−10.10	−10.95	−13.06	−14.53	26.5
2	20	18	1.30	1.07	0.30	0.01	−0.25	131.5
3	20	1	8.08	−2.71	−4.36	−4.91	−5.16	113.3
4	20	0	16.82	−9.83	−11.35	−12.16	−12.45	98.8
5	20	0	29.46	−9.77	−11.92	−13.30	−13.86	97.2
6	20	0	42.76	−9.27	−10.94	−12.62	−13.49	92.0

Table 3.5: Results of the matheuristic for the “random” instances

			0–splitting		**1–splitting**	**3–splitting**	**7–splitting**	
K	♯ 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 3.6: Results of the matheuristic for the “even” instances

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 3.5 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%, respectively. 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 3.5 say that, for the 20 instances with 60 to 127 vertices and 6 drones, the costs of the solutions provided by the matheuristic in 92.0 seconds on average are, without splitting the required lines, 9.27% better than those obtained by the branch and cut in one

				GapUB				
	K	♯ opt	♯ opt M	0–splitting	1–splitting	3–splitting	7–splitting	Time
R	2	37	25	0.86	−0.06	−0.53	−0.94	76.2
E		15	6	1.76	−2.99	−4.27	−5.20	55.8
R	3	13	7	1.35	1.07	−0.38	−0.82	22.6
E		5	4	0.86	−5.30	−6.42	−7.68	15.1
R	4	4	3	0.08	−0.23	−0.91	−4.94	14.9
E		2	0	2.55	−2.36	−2.44	−2.78	10.3
R	5	2	1	3.02	2.82	−1.48	−2.67	10.5
E		1	1	0.00	−2.72	−4.04	−4.30	7.8
R	6	3	3	0.00	−0.44	−1.33	−2.72	7.8
E		1	1	0.00	−6.40	−9.11	−9.38	11.5

Table 3.7: Results for the instances with known optimal solutions

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 values 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 computational results shown in Table 3.6 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 3.7 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 3.8 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

			LS		RO	
Name	$\lvert V\rvert$	$\lvert E\rvert$	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 3.8: Impact of local search and route optimization procedures (instances grouped by size)

"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 3.8 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 is 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 3.9 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.

		LS		**RO**	
K	♯ inst	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 3.9: Impact of local search and route optimization procedures (instances grouped by number of drones)

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.

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

Table 3.10: Analysis of the contribution of the different local search procedures

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 3.10 reports the results obtained. Column 2 shows 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.

Chapter 4

On the length constrained K–RPP and the K–RPP polyhedron

In this chapter, we focus on a deeper theoretical study of the length constrained K–vehicles rural postman problem (LC K–RPP), the *discrete* arc routing problem defined in Chapter 3. As discussed in the previous chapter, each required line of the LC K–DRPP instance can be considered as a chain of required edges of an undirected graph, thus defining an LC K–RPP instance. Clearly, the greater the number of edges each line is divided, the better approximate the LC K–RPP solution will be to the LC K–DRPP one. However, the difficulty of solving to optimality an LC K–RPP instance increases considerably with the size of the instance. It is convenient to reinforce a formulation of the problem with valid inequalities (and even more so if they define a facet of the polyhedron of solutions) that are useful for the development of a sophisticated exact algorithm, capable of solving larger instances.

In Section 3.1, a "directed" formulation for the LC K–RPP, with two integer variables x_{ij}^k and x_{ji}^k for each edge (i, j) and for each drone k, was proposed. We present in Section 4.1 a new formulation for the LC $K-$RPP that inherits from the formulation proposed in Corberán et al. (2013) for the maximum benefit Chinese postman problem (MBCPP) the idea of considering two binary variables for each edge representing the first and second traversal of the edge, respectively.

Throughout this chapter, we will carry out a polyhedral study of the set of solutions of a relaxed formulation, namely the set of solutions of the formulation presented in Section 4.1 but without considering the length constraint on the drone routes. We study the dimension of that resulting polyhedron of solutions, which will be called $K-$RPP, and prove some inequalities from the formulation to induce facets of it. Moreover, we present three families of valid inequalities and some conditions for them to induce facets of the polyhedron: parity inequalities, p–connectivity inequalities, and K–C inequalities. These families of inequalities have been proposed and proved to be facet–inducing inequalities for other polyhedra associated with arc routing problems in the literature, and have been generalized on this work. We first study in Section 4.2.1 the special case $K = 1$ (with a single drone), whose polyhedral study is necessary for adressing the general case $K \geq 2$ in Section 4.2.2. We close this chapter by introducing in Section 4.2.3 other valid inequalities for the LC $K-$RPP based on the length constraint on each drone route.

4.1 A new formulation for the length constrained K–RPP

As described in the previous chapter, the LC $K-$RPP is defined on an undirected multigraph $G = (V, E) = (V, E_R \cup E_{NR})$. The set of vertices V contains the endpoints of the required edges plus the depot, denoted by 1. The set of required edges E_R contains the edges that must be serviced, while the set of non–required edges E_{NR} contains an edge between each pair of vertices in V, i.e., (V, E_{NR}) is a complete graph. Each $e \in E_R$ has an associated service cost $c_e^s \geq 0$ and each non–required edge $e \in E_{NR}$ has an associated deadheading cost $c_e \geq 0$ given by the Euclidean distance computed from its endpoints. Note that each $e \in E_R$ has a parallel non–required edge, which we denote as e', and it is assumed that the cost of traveling while servicing the required edge $e = (i, j)$ is greater than or equal to the cost of flying directly from i to j ($c_e^s \geq c_{e'}$). E'_{NR} represents the set of non–required edges parallel to a required one, while $E''_{NR} = E_{NR} \setminus E'_{NR}$. The goal of the LC $K-$RPP is to find K tours (closed walks starting and ending at the depot) with length no greater than L, that jointly traverse (and service) all the required edges with minimum total cost.

We want to point out that in the LC $K-$RPP instances the non–required edges form a complete graph, whereas in classical arc routing problems the graph often corresponds to a sparse network. To make a more general study, in the following sections we do not assume that the edges in E_{NR} form a complete graph, although we keep the assumption that in E there is an edge e' parallel to each $e \in E_R$.

The formulation we present here for the LC K–RPP, which we will refer to as (F2) to distinguish it from the one presented in Chapter 3, considers a binary variable x_e^k for each edge $e \in E_R$ and for each drone $k \in \{1, \ldots, K\}$, and two binary variables x_e^k and y_e^k for each edge $e \in E_{NR}$ and for each $k \in \{1, \ldots, K\}$. Variable x_e^k for each edge $e \in E_R$ takes the value 1 if e is traversed (and serviced) by drone k, and 0 otherwise. Variables x_e^k and y_e^k for each edge $e \in E_{NR}$ take the value 1 if e is traversed or traversed twice, respectively, by drone k, and 0 otherwise. In other words, variables x_e^k and y_e^k represent the first and second traversal by drone k of the non–required edge e. The use of these variables is inspired by the work in Corberán et al. (2013) for the MBCPP, and we keep their description in Table 4.1.

Table 4.1: Decision variables of the LC $K-$RPP formulation (F2)

x_e^p	1, if edge $e \in E_R$ is traversed and serviced by drone k, and 0 otherwise
x_e^p	1, if edge $e \in E_{NR}$ is traversed by drone k, and 0 otherwise
y_e^p	1, if edge $e \in E_{NR}$ is traversed twice by drone k, and 0 otherwise

Note that, for each drone, there are three variables to represent the traversals between the two endpoints i and j of a required edge e. The reason is that we need to distinguish among traversing e while serving it (with a cost c_e^s) and deadheading e' once or twice (with a cost $c_{e'} \leq c_e^s$). Although in all the optimal solutions the three variables will never be non–zero simultaneously (see Theorem 4.1.1), we need the three variables to state the objective function of the problem formulation.

Theorem 4.1.1. *For each $e \in E_R$ and its parallel edge $e' \in E'_{NR}$, and for each $k \in \{1, \dots, K\}$, the inequality $x_e^k + y_{e'}^k \leq 1$ is satisfied by any optimal solution of the* LC K–RPP *(if $c_{e'} > 0$).*

Proof: If a solution satisfies $x_e^k + y_{e'}^k = 2$ for a drone k and an edge $e = (i, j) \in E_R$, necessarily $x_{e'} = 1$ (because $x_{e'} \geq y_{e'}$ holds) and the drone travels three times between i and j, so we can remove two copies and get a better feasible solution. □

The notation we use next is very similar to the one already used in the previous chapter. Given two subsets of vertices $S, S' \subseteq V$, $(S : S')$ denotes the edge set with one endpoint in S and the other one in S'. Given a subset $S \subseteq V$, we denote $\delta(S) = (S : V \backslash S)$ and $E(S) = (S : S)$. For simplicity, when $S = \{i\}$, $i \in V$, we write $\delta(i)$ instead of $\delta(\{i\})$. For any subset $F \subseteq E = E_R \cup E_{NR}$, we denote $F_R = F \cap E_R$ and $F_{NR} = F \cap E_{NR}$, and we write $\delta_R(S) = \delta(S) \cap E_R$ instead of $\delta(S)_R$.

The LC K–RPP can be formulated here as follows:

$$\text{(F2)} \qquad \text{Minimize} \qquad \sum_{k=1}^{K} \left(\sum_{e \in E_R} c_e^s x_e^k + \sum_{e \in E_{NR}} c_e \big(x_e^k + y_e^k \big) \right)$$

s.t.

$$\sum_{e \in \delta_R(i)} x_e^k + \sum_{e \in \delta_{NR}(i)} \big(x_e^k + y_e^k \big) \equiv 0 \pmod 2, \qquad \forall i \in V, \ \ \forall k = 1, \dots, K \tag{4.1}$$

$$\sum_{e \in \delta_R(S)} x_e^k + \sum_{e \in \delta_{NR}(S)} \big(x_e^k + y_e^k \big) \geq 2 x_f^k, \qquad \forall S \subseteq V \setminus \{1\}, \forall f \in E(S), \forall k \tag{4.2}$$

$$\sum_{e \in E_R} c_e^s x_e^k + \sum_{e \in E_{NR}} c_e \big(x_e^k + y_e^k \big) \leq L, \qquad \forall k = 1, \dots, K \tag{4.3}$$

$$\sum_{k=1}^{K} x_e^k \geq 1, \qquad \forall e \in E_R \tag{4.4}$$

$$x_e^k \geq y_e^k, \qquad \forall e \in E_{NR}, \ \ \forall k = 1, \dots, K \tag{4.5}$$

$$x_e^k \in \{0, 1\}, \qquad \forall e \in E_R, \ \ \forall k = 1, \dots, K \tag{4.6}$$

$$x_e^k, y_e^k \in \{0, 1\}, \qquad \forall e \in E_{NR}, \ \ \forall k = 1, \dots, K \tag{4.7}$$

Constraints (4.1) force each drone to visit each vertex an even number of times, possibly zero. Conditions (4.2) ensure each single route is connected and connected to the depot, while constraints (4.3) guarantee that the length of each route does not exceed L. The traversal of all the required edges is ensured by constraints (4.4). Constraints (4.5) guarantee that a second traversal of a non required edge by a drone can only occur when it has been traversed previously by this drone. Constraints (4.6) and (4.7) are the binary conditions for the variables.

Note that constraints (4.1) are not linear, although they could be linearized by introducing additional general integer variables, z_i^k, as follows

$$\sum_{e\in\delta_R(i)} x_e^k + \sum_{e\in\delta_{NR}(i)} (x_e^k + y_e^k) = 2z_i^k.$$

Instead, as we will see in Section 4.2.2, we will introduce some linear inequalities on the variables x_e^k, y_e^k (parity and K–C inequalities). Note also that the previous formulation allows "not feasible" solutions with isolated subtours of non–required edges, although these solutions are not optimal.

4.2 Polyhedral study of the K–RPP

In this section we study a polyhedron of solutions associated with a relaxation of the proposed formulation for the LC K–RPP. As with other routing problems with several vehicles, determining the dimension of the polyhedron defined as the convex hull of the LC K–RPP solutions is a very difficult task, because it depends also on the edge costs c_e and c_e^s, the number of vehicles K, and the length limit L. Even in some cases, the polyhedron could be empty. However, if we remove the constraints (4.3) that limit the length of each route, the relaxed problem becomes the $K-$vehicles Rural Postman Problem (K–RPP), whose polyhedron can indeed be studied. This is interesting because some of its facets could also be facets of the original LC K–RPP polyhedron, and it is a way of guaranteeing the strength of the constraints in the formulation and of the valid inequalities we can find.

Since the LC $K-$RPP formulation (F2), and therefore the $K-$RPP formulation (i.e., formulation (F2) without constraints (4.3)), is based on the one presented in Corberán et al. (2013) for the MBCPP, some of the inequalities proposed in that article for the MBCPP can be transformed into valid inequalities for the K–RPP (see Theorem 4.2.1). However, to prove that these new inequalities, and the inequalities in the K–RPP formulation, are facet inducing, we will first discuss the polyhedron associated with the K–RPP in the special case when $K = 1$ in Section 4.2.1.

Let a K–RPP *solution* denotes any set of K tours on graph G starting and ending at the depot and jointly servicing (traversing) all the required edges (without considering the lenght constraints (4.3)). Associated with each K–RPP solution we can consider:

(a) K incidence vectors $(x^k, y^k) \in \mathbb{Z}^{2|E_{NR}|+|E_R|}$, one for each tour k, where variables x_e^k take the value 1 if edge e is traversed and variables y_e^k take the value 1 if non–required edge e is traversed twice, and

(b) K support graphs $(V, E^{(x^k,y^k)})$, one for each tour, where $E^{(x^k,y^k)}$ contains one copy of edge $e \in E$ for each variable $x_e^k = 1$ or $y_e^k = 1$.

Note that the support graphs are even and connected. Conversely, any even and connected subgraph of G corresponds to a tour on G. In fact, an incidence vector or a subgraph may correspond to several different closed walks, but all of them have the same cost and they can be easily computed (with the Hierholzer (1873) algorithm, for example). Hence, and for the sake of simplicity, we will let *tour on G* denote the closed walk, its incidence vector, and its corresponding support graph.

In what follows, we study the K–RPP polyhedron defined as the convex hull of the vectors

$$(x^1, y^1, x^2, y^2, \ldots, x^K, y^K) \in \mathbb{Z}^{K(2|E_{NR}|+|E_R|)}$$

corresponding to K–RPP solutions on G, that is, each (x^k, y^k), for $k = 1, \ldots, K$, is a tour on G starting and ending at the depot, and all the required edges are traversed by, at least, one drone. Let $K-$RPP(G) denote this polyhedron. Note that we do not consider inequalities $x_e^k + y_{e'}^k \leq 1$ from Theorem 4.1.1, and we use inequalities (4.4) instead of using $\sum_{k=1}^K x_e^k = 1$. Although the feasible solutions that satisfy $x_e^k + y_{e'}^k = 2$ or $\sum_{k=1}^K x_e^k > 1$ cannot be optimal solutions (if $0 < c_{e'} < c_e^s$), these will be useful in the proofs of the results presented in Section 4.2.2.

To make this polyhedral study, we need some results presented in Corberán et al. (2013) for the MBCPP. Given an undirected connected graph $G = (V, E)$, where $1 \in V$ represents the depot, with two benefits for each edge $e \in E$ associated with the first and the second traversals of e, respectively, the MBCPP consists of finding a tour starting from the depot, traversing some of the edges in E at most twice and returning to the depot, with maximum total benefit. The MBCPP is formulated with two binary variables x_e and y_e for each edge $e \in E$ representing the first and second traversal of e, respectively. It is shown that the convex hull of all the MBCPP tours, i.e., the vectors (x, y) satisfying

$$\sum_{e\in\delta(i)} \big(x_e + y_e\big) \equiv 0 \pmod 2, \qquad \forall i \in V, \tag{4.8}$$

$$\sum_{e\in\delta(S)} \big(x_e + y_e\big) \geq 2x_f, \qquad \forall S \subset V \setminus \{1\}, \;\; \forall f \in E(S), \tag{4.9}$$

$$x_e \geq y_e, \qquad \forall e \in E, \tag{4.10}$$

$$x_e, y_e \in \{0, 1\}, \qquad \forall e \in E, \tag{4.11}$$

is a full dimensional polytope and several families of valid and facet–inducing inequalities are described.

The formulation we present here for the K–RPP has only one variable associated with each required edge, while, if we consider the MBCPP on the same graph, we have two variables for each edge, including the required ones. Nevertheless, given a K–RPP solution

$$(x^1, y^1, x^2, y^2, \ldots, x^K, y^K) \in \mathbb{Z}^{K(2|E_{NR}|+|E_R|)},$$

each single route (x^k, y^k) is a closed walk starting and ending at the depot, and it can be completed with variables $y_e = 0$ for each $e \in E_R$ to obtain a MBCPP tour. Hence we have the following theorem:

Theorem 4.2.1. *Let $f(x, y) \geq \alpha$ be a valid inequality for the* MBCPP *on graph G. By removing all the variables y_e, $e \in E_R$, the resulting inequality $f(x^k, y^k) \geq \alpha$ is valid for the $K-$*RPP*, for each drone $k \in \{1, \ldots, K\}$.*

For example, we obtain inequalities (4.2) from inequalities (4.9) of the MBCPP formulation. Furthermore, from several families of valid inequalities for the MBCPP, namely parity, $p-$connectivity, and K–C inequalities, we will obtain valid inequalities for the $K-$RPP in Section 4.2.2. Besides some results from the MBCPP, we need some polyhedral results from the "1–RPP", the version of the K–RPP with only one vehicle ($K = 1$). This is an auxiliary problem which really does not make sense here since the K–RPP is a multi–vehicle problem, but whose polyhedral study, carried out in the next section, will be used in Section 4.2.2 for the polyhedral study of the general case with $K \geq 2$.

4.2.1 The 1–RPP polyhedron

The K–RPP for $K = 1$, which we will call 1–RPP, is the well known RPP but with some special features. First, it is defined on a graph that has a non–required edge parallel to each required one. Second, the problem is formulated with three variables associated with the traversal of a required edge e and its parallel non–required one e'. Note that for the 1–RPP, $x_e = 1$ holds for each $e \in E_R$ and, from Theorem 4.1.1, we obtain that $y_{e'} = 0$ for each $e' \in E'_{NR}$ in all the optimal 1–RPP tours. Hence, these variables could be removed from the formulation. However, since this is not true for $K > 1$, we will keep these variables because they are necessary in the proofs of the polyhedral study of the K–RPP(G) for $K > 1$. Hence, we will accept feasible (but not optimal) solutions with some variables $y_{e'} = 1$.

On the other hand, although it is natural in problems with drones to assume that all the vertices (except maybe the depot) are incident with required edges, we will not consider here this assumption to make a more general study. Therefore, we consider in this section an undirected and connected graph $G = (V, E)$, with a set $E_R \subset E$ of required edges, and where the set V_R, formed with the vertices incident with some edge in E_R plus the depot, is not necessarily equal to V. We assume $E = E_R \cup E'_{NR} \cup E''_{NR}$, where E'_{NR} is the set of non–required edges parallel to an edge in E_R. In the same way, we will not assume that graph (V, E_{NR}) is complete.

Let a 1–RPP tour denotes a closed walk on graph G starting and ending at the depot, and servicing all the required edges. As before, we will use 1–RPP tour also to denote its incidence vector $(x, y) \in \mathbb{Z}^{2|E_{NR}|+|E_R|}$ and its support graph. The polyhedron 1–RPP(G) is defined as the convex hull of all the 1–RPP tours in G. Note that the set of constraints of the K–RPP formulation, adapted to the case $K = 1$, is:

$$\sum_{e\in\delta_R(i)} x_e + \sum_{e\in\delta_{NR}(i)} \big(x_e + y_e\big) \equiv 0 \pmod 2, \qquad \forall i\in V, \tag{4.12}$$

$$\sum_{e\in\delta_R(S)} x_e + \sum_{e\in\delta_{NR}(S)} \big(x_e + y_e\big) \geq 2x_f, \qquad \forall S\subseteq V\setminus\{1\}, \forall f\in E(S), \tag{4.13}$$

$$x_e = 1, \qquad \forall e \in E_R, \tag{4.14}$$

$$x_e \geq y_e, \qquad \forall e\in E_{NR}, \tag{4.15}$$

$$x_e, y_e \in \{0,1\}, \qquad \forall e\in E_{NR}. \tag{4.16}$$

In what follows, we will obtain the dimension of 1–RPP(G), and we will also study some conditions under which some of the above constraints of the 1–RPP formulation, as well as other families of valid inequalities presented, define facets of it. For this study, we need to build several 1–RPP tours in graph G. An example of 1–RPP tour is given by the graph formed with two copies of each edge in E_{NR} and then replacing one copy of each $e \in E'_{NR}$ by the required edge parallel to e. This basic tour will be used in the proof of Theorem 4.2.2. More specific and detailed 1–RPP tours will be built for the proofs of other next theorems. In order to do this, we need to introduce some definitions.

Consider the (generally disconnected) subgraph (V_R, E_R) of G. Each connected component of this subgraph is called an *R–connected* component of G. An R–connected component may consist only of the depot. A vertex of G that does not belong to any R–connected component (i.e., which is not incident with any required edge) is called an *isolated* vertex.

Given a subset of vertices $V_o \subseteq V$, with $|V_o|$ even, a subset of edges $M \subseteq E$ is a T–*join* if the degree of a vertex v in the subgraph (V, M) is odd if, and only if, $v \in V_o$. It is known that, if G is connected, it has a T–join for each set $V_o \subseteq V$, with $|V_o|$ even (see Nemhauser and Wolsey, 1988 for instance). Given $G = (V, E) = (V, E_R \cup E'_{NR} \cup E''_{NR})$, let $V_o^R \subseteq V$ be the set of R–odd vertices, i.e., vertices incident with an odd number of required edges. Let $M \subseteq E_{NR}$ be any corresponding T–join. The set of edges $M \cup E_R$ form an even graph, although not necessarily connected. If we add the edges in a closed walk starting at the depot, visiting at least one node in each connected component of $M \cup E_R$ and ending at the depot, we obtain a 1–RPP tour.

The dimension of the polyhedron defined as the convex hull of all the 1–RPP tours is given in the following theorem:

Theorem 4.2.2. *dim(1–RPP(G)) $= 2|E_{NR}|$ if, and only if, (V, E_{NR}) is a 3–edge connected graph.*

Proof: 1–RPP(G) is a polytope in $\mathbb{R}^{2|E_{NR}|+|E_R|}$. Since all its points satisfy equations (4.14), which are linearly independent, we have dim(1–RPP(G)) $\leq 2|E_{NR}|$. If (V, E_{NR}) is not 3–edge connected, there is a cut–set $\delta(S)$ with at most 2 non–required edges. If $\delta(S)$ contains exactly two edges, namely e and f, all the 1–RPP tours satisfy the equation $x_e - y_e = x_f - y_f$. If $\delta(S) = \{e\}$, all the 1–RPP tours satisfy $x_e = y_e$. Given that these equations are linearly independent from equations (4.14), we have dim(1–RPP(G)) $< 2|E_{NR}|$.

On the other hand, let us now suppose that graph (V, E_{NR}) is 3–edge connected. We will prove that dim(1–RPP(G)) $\geq 2|E_{NR}|$. Let $ax + by = c$, i.e.,

$$\sum_{e \in E_R} a_e x_e + \sum_{e \in E_{NR}} a_e x_e + \sum_{e \in E_{NR}} b_e y_e = c \tag{4.17}$$

be an equation satisfied by all the 1–RPP tours. We have to prove that this equation is a linear combination of equations (4.14), i.e., to prove that

$$\begin{aligned} a_e &= 0, \ \ \forall e \in E_{NR}, \\ b_e &= 0, \ \ \forall e \in E_{NR}, \text{ and} \\ c &= \sum_{e \in E_R} a_e. \end{aligned}$$

Let $T = (x, y)$ be the 1–RPP tour formed with two copies of each edge in E_{NR} and then replacing one copy of each $e \in E'_{NR}$ by the required edge parallel to e. All the entries of T are equal to 1 except for $y_e = 0, \forall e \in E'_{NR}$ and, by substituting it in (4.17), we obtain

$$\sum_{e \in E_R} a_e + \sum_{e \in E_{NR}} a_e + \sum_{e \in E''_{NR}} b_e = c. \tag{4.18}$$

Let be $f \in E''_{NR}$. Since (V, E_{NR}) is a 3–edge connected graph, the tour T^{-2f} obtained after removing from T the two copies of f is also a 1–RPP tour. Hence, by substituting it in (4.17), we obtain

$$\sum_{e \in E_R} a_e + \sum_{e \in E_{NR} \setminus \{f\}} a_e + \sum_{e \in E''_{NR} \setminus \{f\}} b_e = c,$$

and by subtracting this equation from (4.18), we obtain $a_f + b_f = 0$ for all $f \in E''_{NR}$.

Let $\mathcal{C}$ be an arbitrary cycle on G formed by non–required edges, and let us denote $\mathcal{C}' = \mathcal{C} \cap E'_{NR}$ and $\mathcal{C}'' = \mathcal{C} \cap E''_{NR}$. If we remove from T one copy of each edge in $\mathcal{C}$, we obtain another 1–RPP tour $T - \mathcal{C}$. After substituting it in (4.17) and subtracting the corresponding equation from (4.18), we obtain $a(\mathcal{C}') + b(\mathcal{C}'') = 0$. On the other hand, if we add to T one copy of each edge in $\mathcal{C}$ and, then, we remove two copies of each edge appearing three times, we obtain another 1–RPP tour $T + \mathcal{C}$. After substituting it in (4.17) and subtracting the corresponding equation from (4.18), we obtain $b(\mathcal{C}') - b(\mathcal{C}'') = 0$, and, as $a_f + b_f = 0$ for all $f \in E''_{NR}$, $b(\mathcal{C}') + a(\mathcal{C}'') = 0$. Hence, for each cycle $\mathcal{C} = \mathcal{C}' \cup \mathcal{C}''$ on G formed by non–required edges,

$$a(\mathcal{C}') + b(\mathcal{C}'') = 0, \tag{4.19}$$
$$b(\mathcal{C}') + a(\mathcal{C}'') = 0. \tag{4.20}$$

Let $f = (i, j) \in E''_{NR}$. Since (V, E_{NR}) is a 3–edge connected graph, there are two edge–disjoint paths $\mathcal{P}_1$, $\mathcal{P}_2$ joining vertices i and j with non–required edges different from f. Consider the three cycles $\mathcal{P}_1 \cup \{f\}$, $\mathcal{P}_2 \cup \{f\}$, and $\mathcal{P}_1 \cup \mathcal{P}_2$, for which equation (4.19) holds. Hence, we have

$$\begin{aligned} a(\mathcal{P}'_1) + b(\mathcal{P}''_1) + b_f = 0, \\ a(\mathcal{P}'_2) + b(\mathcal{P}''_2) + b_f = 0, \\ a(\mathcal{P}'_1) + b(\mathcal{P}''_1) + a(\mathcal{P}'_2) + b(\mathcal{P}''_2) = 0, \end{aligned}$$

and we obtain $b_f = 0$. As $a_f + b_f = 0$ holds, we have $a_f = b_f = 0$ for all $f \in E''_{NR}$.

Let $f = (i, j) \in E'_{NR}$. Let $\mathcal{P}_1$, $\mathcal{P}_2$ two edge–disjoint paths as above and consider the three cycles $\mathcal{P}_1 \cup \{f\}$, $\mathcal{P}_2 \cup \{f\}$, and $\mathcal{P}_1 \cup \mathcal{P}_2$. From equation (4.19), we have

$$\begin{aligned} a(\mathcal{P}'_1) + b(\mathcal{P}''_1) + a_f &= 0, \\ a(\mathcal{P}'_2) + b(\mathcal{P}''_2) + a_f &= 0, \\ a(\mathcal{P}'_1) + b(\mathcal{P}''_1) + a(\mathcal{P}'_2) + b(\mathcal{P}''_2) &= 0, \end{aligned}$$

from which we obtain $a_f = 0$, and from (4.20), we also have

$$b(\mathcal{P}'_1) + a(\mathcal{P}''_1) + b_f = 0,$$

$$\begin{aligned} b(\mathcal{P}_2') + a(\mathcal{P}_2'') + b_f &= 0, \\ b(\mathcal{P}_1') + a(\mathcal{P}_1'') + b(\mathcal{P}_2') + a(\mathcal{P}_2'') &= 0, \end{aligned}$$

from which we obtain $b_f = 0$. Hence, we have $a_f = b_f = 0$ for all $f \in E_{NR}'$, and then, $a_e = b_e = 0$ for all $e \in E_{NR}$. By substituting in (4.18), we obtain $\sum_{e\in E_R} a_e = c$ and we are done. □

We will assume in what follows that (V, E_{NR}) is a 3–edge connected graph.

Facet–inducing inequalities obtained from the formulation

Let us first prove some inequalities from the formulation to be facet–defining of the 1–RPP polyhedron: the trivial inequalities given by the bounds of the variables, inequalities (4.15), and connectivity inequalities (4.13).

Theorem 4.2.3. *Inequality $y_e \geq 0$, for each edge $e \in E_{NR}$, is facet–inducing of* 1–RPP(G).

Proof: Let $ax + by \geq c$, i.e.,

$$\sum_{f\in E_R} a_f x_f + \sum_{f\in E_{NR}} a_f x_f + \sum_{f\in E_{NR}} b_f y_f \geq c,$$

be a valid inequality such that

$$\{(x,y) \in 1\text{–RPP}(G): \quad y_e = 0\} \subseteq \{(x,y) \in 1\text{–RPP}(G): \quad ax + by = c\}.$$

We have to prove that this inequality is a linear combination of the equalities (4.14) and $y_e \geq 0$. Note that this means to prove that

$$\begin{aligned} a_f &= 0, \;\; \forall f \in E_{NR}, \\ b_f &= 0, \;\; \forall f \in E_{NR}, \; f \neq e, \text{ and} \\ c &= \sum_{f\in E_R} a_f. \end{aligned}$$

i) We will first prove it for $e \in E_{NR}''$. Consider the tour T^{-2e} of the proof of Theorem 4.2.2, which satisfies $y_e = 0$ and, hence, $ax + by = c$ holds. By replacing the incidence vector in $ax + by = c$, we obtain

$$\sum_{f\in E_R} a_f + \sum_{f\in E_{NR}\setminus\{e\}} a_f + \sum_{f\in E_{NR}''\setminus\{e\}} b_f = c. \tag{4.21}$$

Let be $g \in E_{NR}''$, $g \neq e$. Since (V, E_{NR}) is a 3–edge connected graph, the tour T^{-2e-2g} obtained after removing from T^{-2e} the two copies of g is also a 1–RPP tour satisfying $y_e = 0$. Hence, by substituting it in $ax + by = c$ and by subtracting the corresponding equality from (4.21), we obtain that $a_g + b_g = 0$, $\forall g \in E_{NR}''$, $g \neq e$.

Let be $g \in E'_{NR}$. Since (V, E_{NR}) is a 3–edge connected graph, $(V, E_{NR} \setminus \{e, g\})$ is a connected graph, and there is a T–join in $(V, E_{NR} \setminus \{e, g\})$ connecting the R–odd vertices. The 1–RPP tour T^* build wit this T–join does not traverse g and satisfies $y_e = 0$. Furthermore, the tour obtained after adding to T^* the two copies of g is also a 1–RPP tour satisfying $y_e = 0$. Hence, by substituting them in $ax + by = c$ and by subtracting the corresponding equalities, we obtain that $a_g + b_g = 0,\ \forall g \in E'_{NR}$.

For each cycle $\mathcal{C} = \mathcal{C}' \cup \mathcal{C}''$ on G formed by non–required edges, the tour $T^{-2e} + \mathcal{C}$ obtained after adding to T^{-2e} one copy of each edge in $\mathcal{C}$ (and, then, removing two copies of any edge traversed three times) is also a 1–RPP tour satisfying $y_e = 0$. Proceeding as in the proof of Theorem 4.2.2, we obtain $b(\mathcal{C}') - b(\mathcal{C}'') = 0$ and, given that $a_g + b_g = 0,\ \forall g \in E''_{NR},\ g \neq e$, we obtain that $b(\mathcal{C}') + a(\mathcal{C}'' \setminus \{e\}) - b_e = 0$ if $e \in \mathcal{C}$, and $b(\mathcal{C}') + a(\mathcal{C}'') = 0$ if $e \notin \mathcal{C}$.

Let be $f = (i, j) \in E''_{NR}$. Since (V, E_{NR}) is a 3–edge connected graph, there are two edge–disjoint paths $\mathcal{P}_1$, $\mathcal{P}_2$ joining vertices i and j with non–required edges different from f. Let us assume, for instance, that $e \in \mathcal{P}_1$. Consider the two cycles $\mathcal{P}_1 \cup \{f\}$ and $\mathcal{P}_1 \cup \mathcal{P}_2$, for which equation $b(\mathcal{C}') + a(\mathcal{C}'' \setminus \{e\}) - b_e = 0$ holds, and the cycle $\mathcal{P}_2 \cup \{f\}$, for which equation $b(\mathcal{C}') + a(\mathcal{C}'') = 0$ holds. Hence, we have:

$$\begin{aligned} b(\mathcal{P}'_1) + a(\mathcal{P}''_1 \setminus \{e\}) - b_e + a_f &= 0, \\ b(\mathcal{P}'_1) + a(\mathcal{P}''_1 \setminus \{e\}) - b_e + b(\mathcal{P}'_2) + a(\mathcal{P}''_2) &= 0, \text{ and} \\ b(\mathcal{P}'_2) + a(\mathcal{P}''_2) + a_f &= 0, \end{aligned}$$

and we obtain $a_f = 0$. Furthermore, if $f \neq e$, $a_f + b_f = 0$ holds, we have $b_f = 0$ for all $f \in E''_{NR} \setminus \{e\}$. The case $e \notin \mathcal{P}_1 \cup \mathcal{P}_2$ is similar.

Let be $f = (i, j) \in E'_{NR}$. Let $\mathcal{P}_1$, $\mathcal{P}_2$ be two edge–disjoint paths as above and assume in this case that $e \notin \mathcal{P}_1$ and $e \notin \mathcal{P}_2$. Consider the three cycles $\mathcal{P}_1 \cup \{f\}$, $\mathcal{P}_2 \cup \{f\}$, and $\mathcal{P}_1 \cup \mathcal{P}_2$. Now we have:

$$\begin{aligned} b(\mathcal{P}'_1) + a(\mathcal{P}''_1) + b_f &= 0, \\ b(\mathcal{P}'_2) + a(\mathcal{P}''_2) + b_f &= 0, \text{ and} \\ b(\mathcal{P}'_1) + a(\mathcal{P}''_1) + b(\mathcal{P}'_2) + a(\mathcal{P}''_2) &= 0, \end{aligned}$$

from which we obtain $b_f = 0$. Furthermore, as $a_f + b_f = 0,\ \forall f \in E'_{NR}$ we obtain that $a_f = 0$. Hence, $a_f = b_f = 0$ for all $f \in E_{NR} \setminus \{e\}$ and $a_e = 0$ and, by replacing them in (4.21) we obtain $\sum_{f \in E_R} a_f = c$, and we are done.

ii) We will prove it now for $e \in E'_{NR}$. The tour T of the proof of Theorem 4.2.2 satisfies $y_e = 0$ and, hence, $ax + by = c$ holds, and by replacing the incidence vector, we obtain equation:

$$\sum_{e \in E_R} a_e + \sum_{e \in E_{NR}} a_e + \sum_{e \in E''_{NR}} b_e = c. \tag{4.22}$$

Let be $f \in E''_{NR}$. Since (V, E_{NR}) is a 3–edge connected graph, the tour T^{-2f} is also a 1–RPP tour satisfying $y_e = 0$. Hence, by substituting it in $ax + by = c$ and by subtracting the corresponding equality from (4.22) we obtain that $a_f + b_f = 0,\ \forall f \in E''_{NR}$.

Let be $f \in E'_{NR}$, $f \neq e$. Since (V, E_{NR}) is a 3–edge connected graph, $(V, E_{NR} \setminus \{e, f\})$ is a connected graph, and there is a T–join in $(V, E_{NR} \setminus \{e, f\})$ connecting the R–odd vertices. The 1–RPP tour T^* build wit this T–join does not traverses f and satisfies $y_e = 0$. Furthermore, the tour obtained after adding to T^* the two copies of f is also a 1–RPP tour satisfying $y_e = 0$. Hence, by subtracting the corresponding equalities, we obtain that $a_f + b_f = 0,\ \forall f \in E'_{NR},\ f \neq e$.

For each cycle $\mathcal{C} = \mathcal{C}' \cup \mathcal{C}''$ on G formed by non–required edges, the tour $T - \mathcal{C}$ obtained after subtracting from T one copy of each edge in $\mathcal{C}$ is also a 1–RPP tour satisfying $y_e = 0$. Proceeding as in the proof of Theorem 4.2.2, we obtain $a(\mathcal{C}') + b(\mathcal{C}'') = 0$.

Let be $f = (i, j) \in E''_{NR}$. Since (V, E_{NR}) is a 3–edge connected graph, there are two edge–disjoint paths $\mathcal{P}_1$, $\mathcal{P}_2$ joining vertices i and j with non–required edges different from f. Considering the three cycles as in i) we obtain $b_f = 0$ and, as $a_f + b_f = 0$ holds, we have $a_f = b_f = 0$ for all $f \in E''_{NR}$.

Let be $f = (i, j) \in E'_{NR}$. Proceeding as above we obtain $a_f = 0$ and, hence, also $b_f = 0\ \forall f \in E'_{NR} \setminus \{e\}$. We have $a_f = b_f = 0$ for all $f \in E_{NR} \setminus \{e\}$ and $a_e = 0$ and, by replacing them in (4.22) we obtain $\sum_{f \in E_R} a_f = c$, and we are done. □

Theorem 4.2.4. *Inequality $x_e \leq 1$, for each edge $e \in E_{NR}$, is facet–inducing for* 1–RPP(G).

Proof: Let $ax + by \leq c$, i.e.,

$$\sum_{f \in E_R} a_f x_f + \sum_{f \in E_{NR}} a_f x_f + \sum_{f \in E_{NR}} b_f y_f \leq c,$$

be a valid inequality such that

$$\{(x, y) \in 1\text{–RPP}(G) : \quad x_e = 1\} \ \subseteq \ \{(x, y) \in 1\text{–RPP}(G) : \quad ax + by = c\}.$$

We have to prove that this inequality is a linear combination of the equalities (4.14) and $x_e \leq 1$. Note that this means to prove that

$$\begin{aligned} a_f &= 0, \ \forall f \in E_{NR},\ f \neq e, \\ b_f &= 0, \ \forall f \in E_{NR}, \\ c &= \sum_{f \in E_R} a_f + a_e. \end{aligned}$$

i) We will first prove it for $e \in E''_{NR}$. Consider the tour T of the proof of Theorem 4.2.2, which satisfies $x_e = 1$ and, hence, $ax + by = c$ holds. By replacing the incidence vector in $ax + by = c$, we obtain

$$\sum_{f \in E_R} a_f + \sum_{f \in E_{NR}} a_f + \sum_{f \in E''_{NR}} b_f = c. \tag{4.23}$$

Let be $f \in E''_{NR}$, $f \neq e$. Since (V, E_{NR}) is a 3–edge connected graph, the tour T^{-2f} is also a 1–RPP tour satisfying $x_e = 1$. Hence, by substituting it in $ax + by \leq c$ and by subtracting the corresponding equality from (4.23) we obtain that $a_f + b_f = 0,\ \forall f \in E''_{NR},\ f \neq e$.

Let be $f \in E'_{NR}$. Since (V, E_{NR}) is a 3–edge connected graph, $(V, E_{NR} \setminus \{e, f\})$ is a connected graph, and there is a T–join in $(V, E_{NR} \setminus \{e, f\})$ connecting the R–odd vertices. The 1–RPP tour T^* build wit this T–join does not traverse f nor e. T^{*+2e} is also a 1–RPP tour and satisfies $x_e = 1$. Furthermore, $T^{*+2e+2f}$ is also a 1–RPP tour satisfying $x_e = 1$. Hence, by substituting them in $ax + by = c$ and by subtracting the corresponding equalities we obtain that $a_f + b_f = 0$, $\forall f \in E'_{NR}$.

For each cycle $\mathcal{C} = \mathcal{C}' \cup \mathcal{C}''$ on G formed by non–required edges, the tour $T - \mathcal{C}$ obtained after removing from T one copy of each edge in $\mathcal{C}$ is also a 1–RPP tour satisfying $x_e = 1$. Proceeding as in the proof of Theorem 4.2.2 we obtain $a(\mathcal{C}') + b(\mathcal{C}'') = 0$.

Let be $f = (i, j) \in E''_{NR}$. Since (V, E_{NR}) is a 3–edge connected graph, there are two edge–disjoint paths $\mathcal{P}_1$, $\mathcal{P}_2$ joining vertices i and j with non–required edges different from f. Consider the three cycles $\mathcal{P}_1 \cup \{f\}$, $\mathcal{P}_2 \cup \{f\}$, and $\mathcal{P}_1 \cup \mathcal{P}_2$, for which equation $a(\mathcal{C}') + b(\mathcal{C}'') = 0$ holds. Hence, we have:

$$\begin{aligned} a(\mathcal{P}'_1) + b(\mathcal{P}''_1) + b_f &= 0, \\ a(\mathcal{P}'_2) + b(\mathcal{P}''_2) + b_f &= 0, \text{ and} \\ a(\mathcal{P}'_1) + b(\mathcal{P}''_1) + a(\mathcal{P}'_2) + b(\mathcal{P}''_2) &= 0, \end{aligned}$$

and we obtain $b_f = 0$. Furthermore, if $f \neq e$, $a_f + b_f = 0$ holds, we have $a_f = 0$ for all $f \in E''_{NR} \setminus \{e\}$.

Let be $f = (i, j) \in E'_{NR}$. Let $\mathcal{P}_1$, $\mathcal{P}_2$ be two edge–disjoint paths as above and consider the three cycles $\mathcal{P}_1 \cup \{f\}$, $\mathcal{P}_2 \cup \{f\}$, and $\mathcal{P}_1 \cup \mathcal{P}_2$. Now we have:

$$\begin{aligned} a(\mathcal{P}'_1) + b(\mathcal{P}''_1) + a_f &= 0, \\ a(\mathcal{P}'_2) + b(\mathcal{P}''_2) + a_f &= 0, \text{ and} \\ a(\mathcal{P}'_1) + b(\mathcal{P}''_1) + a(\mathcal{P}'_2) + b(\mathcal{P}''_2) &= 0, \end{aligned}$$

from which we obtain $a_f = 0$. Furthermore, as $a_f + b_f = 0$, $\forall f \in E'_{NR}$ we obtain that $b_f = 0$.

Hence, $a_f = b_f = 0$ for all $f \in E_{NR} \setminus \{e\}$ and $b_e = 0$ and, by replacing them in (4.23), we obtain $\sum_{f \in E_R} a_f + a_e = c$ and we are done.

ii) We will prove it now for $e \in E'_{NR}$. Consider the tour T of the proof of Theorem 4.2.2, which satisfies $x_e = 1$ and, hence, $ax + by = c$ holds. By replacing the incidence vector in $ax + by = c$ we obtain

$$\sum_{f \in E_R} a_f + \sum_{f \in E_{NR}} a_f + \sum_{f \in E''_{NR}} b_f = c. \tag{4.24}$$

Let be $f \in E''_{NR}$. Since (V, E_{NR}) is a 3–edge connected graph, the tour T^{-2f} obtained after removing from T the two copies of f is also a 1–RPP tour satisfying $x_e = 1$. Hence, by substituting it in $ax + by = c$ and by subtracting the corresponding equality from (4.24) we obtain that $a_f + b_f = 0$, $\forall f \in E''_{NR}$.

Let be $f \in E'_{NR}$, $f \neq e$. Since (V, E_{NR}) is a 3–edge connected graph, $(V, E_{NR} \setminus \{e, f\})$ is a connected graph, and there is a T–join in $(V, E_{NR} \setminus \{e, f\})$ connecting the R–odd vertices. The 1–RPP tour T^* build wit this T–join does not traverse f nor e. T^{*+2e} is also a 1–RPP tour and satisfies $x_e = 1$. Furthermore, $T^{*+2e+2f}$ is also a 1–RPP tour satisfying $x_e = 1$. Hence, by substituting them in $ax + by = c$ and by subtracting the corresponding equalities we obtain that $a_f + b_f = 0$, $\forall f \in E'_{NR}$, $f \neq e$.

For each cycle $\mathcal{C} = \mathcal{C}' \cup \mathcal{C}''$ on G formed by non–required edges, the tour $T + \mathcal{C}$ obtained after adding to T one copy of each edge in $\mathcal{C}$ (and, then, removing two copies of any edge traversed three times) is also a 1–RPP tour satisfying $x_e = 1$. Proceeding as in the proof of Theorem 4.2.2 we obtain $b(\mathcal{C}') - b(\mathcal{C}'') = 0$ and, given that $a_f + b_f = 0$, $\forall f \in E''_{NR}$, we obtain that $b(\mathcal{C}') + a(\mathcal{C}'') = 0$.

Let be $f = (i,j) \in E''_{NR}$. Since (V, E_{NR}) is a 3–edge connected graph, there are two edge-disjoint paths $\mathcal{P}_1$, $\mathcal{P}_2$ joining vertices i and j with non–required edges different from f. Consider the three cycles $\mathcal{P}_1 \cup \{f\}$, $\mathcal{P}_2 \cup \{f\}$, and $\mathcal{P}_1 \cup \mathcal{P}_2$, for which equation $b(\mathcal{C}') + a(\mathcal{C}'') = 0$ holds. Hence, we have:

$$\begin{aligned} b(\mathcal{P}'_1) + a(\mathcal{P}''_1) + a_f &= 0, \\ b(\mathcal{P}'_2) + a(\mathcal{P}''_2) + a_f &= 0, \text{ and} \\ b(\mathcal{P}'_1) + a(\mathcal{P}''_1) + b(\mathcal{P}'_2) + a(\mathcal{P}''_2) &= 0, \end{aligned}$$

and we obtain $a_f = 0$. Furthermore, as $a_f + b_f = 0$ holds, we have $a_f = 0$ for all $f \in E''_{NR}$.

Let be $f = (i,j) \in E'_{NR}$. Let $\mathcal{P}_1$, $\mathcal{P}_2$ be two edge–disjoint paths as above and consider the three cycles $\mathcal{P}_1 \cup \{f\}$, $\mathcal{P}_2 \cup \{f\}$, and $\mathcal{P}_1 \cup \mathcal{P}_2$. Now we have:

$$\begin{aligned} b(\mathcal{P}'_1) + a(\mathcal{P}''_1) + b_f &= 0, \\ b(\mathcal{P}'_2) + a(\mathcal{P}''_2) + b_f &= 0, \text{ and} \\ b(\mathcal{P}'_1) + a(\mathcal{P}''_1) + b(\mathcal{P}'_2) + a(\mathcal{P}''_2) &= 0, \end{aligned}$$

from which we obtain $b_f = 0$. Furthermore, if $f \neq e$ as $a_f + b_f = 0$, $\forall f \in E'_{NR} \setminus \{e\}$ we obtain that $a_f = 0$. Hence, $a_f = b_f = 0$ for all $f \in E_{NR} \setminus \{e\}$ and $b_e = 0$ and, by replacing them in (4.24), we obtain $\sum_{f \in E_R} a_f + a_e = c$ and we are done. □

Theorem 4.2.5. *Inequalities $x_e \geq y_e$, for each edge $e \in E_{NR}$, are facet–inducing for* 1–RPP(G) *if graph $(V, E_{NR} \setminus \{e\})$ is 3–edge connected.*

Proof: Let $ax + by \geq c$, i.e.,

$$\sum_{f \in E_R} a_f x_f + \sum_{f \in E_{NR}} a_f x_f + \sum_{f \in E_{NR}} b_f y_f \geq c,$$

be a valid inequality such that

$$\{(x,y) \in 1\text{–RPP}(G) : \; x_e = y_e\} \subseteq \{(x,y) \in 1\text{–RPP}(G) : \; ax + by = c\}.$$

We have to prove that this inequality is a linear combination of the equalities (4.14) and $x_e - y_e \geq 0$. Note that this means to prove that

$$\begin{aligned} a_f = b_f &= 0, \; \forall f \in E_{NR}, \; f \neq e, \\ a_e &= -b_e, \\ c &= \sum_{f \in E_R} a_f. \end{aligned}$$

i) We will first prove it for $e \in E''_{NR}$. Consider the tour T of the proof of Theorem 4.2.2, which satisfies $x_e = y_e(= 1)$ and, hence, $ax + by = c$ holds. By replacing the incidence vector in $ax + by = c$, we obtain

$$\sum_{f \in E_R} a_f + \sum_{f \in E_{NR}} a_f + \sum_{f \in E''_{NR}} b_f = c. \tag{4.25}$$

Let be $f \in E''_{NR}$ (including $f = e$). Since (V, E_{NR}) is a 3–edge connected graph, vector T^{-2f} is also a 1–RPP tour and it satisfies $x_e = y_e$. Hence, by replacing it in $ax+by = c$ and then subtracting the corresponding equation from (4.25) we obtain that $a_f + b_f = 0$ for all $f = (i, j) \in E''_{NR}$.

Let be $f \in E'_{NR}$. Since (V, E_{NR}) is a 3–edge connected graph, $(V, E_{NR} \setminus \{e, f\})$ is a connected graph, and there is a T–join in $(V, E_{NR}\setminus\{e, f\})$ connecting the R–odd vertices. The 1–RPP tour T^* build wit this T–join does not traverse f nor e and then satisfies $x_e = y_e$. T^{*+2f} is also a 1–RPP satisfying $x_e = y_e$. By substituting them in $ax + by = c$ and by subtracting the corresponding equalities we obtain that $a_f + b_f = 0$, $\forall f \in E'_{NR}$.

For each cycle $\mathcal{C} = \mathcal{C}' \cup \mathcal{C}''$ on $G \setminus \{e\}$ formed by non–required edges, the tour $T - \mathcal{C}$ obtained after removing from T one copy of each edge in $\mathcal{C}$ is also a 1–RPP tour satisfying $x_e = y_e$. Proceeding as in the proof of Theorem 4.2.2, we obtain $a(\mathcal{C}') + b(\mathcal{C}'') = 0$.

Let be $f = (i, j) \in E''_{NR}$, $f \neq e$. Since $(V, E_{NR} \setminus \{e\})$ is a 3–edge connected graph, there are two edge–disjoint paths $\mathcal{P}_1$, $\mathcal{P}_2$ joining vertices i and j with non–required edges different from e and also different from f. Consider the three cycles $\mathcal{P}_1 \cup \{f\}$, $\mathcal{P}_2 \cup \{f\}$, and $\mathcal{P}_1 \cup \mathcal{P}_2$, for which equation $a(\mathcal{C}') + b(\mathcal{C}'') = 0$ holds. Hence, we have:

$$\begin{aligned}
a(\mathcal{P}'_1) + b(\mathcal{P}''_1) + b_f &= 0, \\
a(\mathcal{P}'_2) + b(\mathcal{P}''_2) + b_f &= 0, \text{ and} \\
a(\mathcal{P}'_1) + b(\mathcal{P}''_1) + a(\mathcal{P}'_2) + b(\mathcal{P}''_2) &= 0,
\end{aligned}$$

and we obtain $b_f = 0$. Furthermore, as $a_f + b_f = 0$ holds, we have $a_f = 0$ for all $f \in E''_{NR} \setminus \{e\}$.

Let be $f = (i, j) \in E'_{NR}$. Let $\mathcal{P}_1$, $\mathcal{P}_2$ be two edge–disjoint paths as above and consider the three cycles $\mathcal{P}_1 \cup \{f\}$, $\mathcal{P}_2 \cup \{f\}$, and $\mathcal{P}_1 \cup \mathcal{P}_2$. Now we have:

$$\begin{aligned}
a(\mathcal{P}'_1) + b(\mathcal{P}''_1) + a_f &= 0, \\
a(\mathcal{P}'_2) + b(\mathcal{P}''_2) + a_f &= 0, \text{ and} \\
a(\mathcal{P}'_1) + b(\mathcal{P}''_1) + a(\mathcal{P}'_2) + b(\mathcal{P}''_2) &= 0,
\end{aligned}$$

from which we obtain $a_f = 0$ and, as $a_f + b_f = 0$ holds, we have $b_f = 0$ for all $f \in E'_{NR}$.

Hence, $a_f = b_f = 0$ for all $f \in E_{NR} \setminus \{e\}$ and $a_e + b_e = 0$ and, by replacing them in (4.25), we obtain $\sum_{f \in E_R} a_f = c$ and we are done.

ii) We will prove it now for $e \in E'_{NR}$. Since (V, E_{NR}) is a 3–edge connected graph, $(V, E_{NR} \setminus \{e\})$ is a connected graph, and there is a T–join in $(V, E_{NR} \setminus \{e\})$ connecting the R–odd vertices. The 1–RPP tour T^* build with this T–join does not traverse e and then satisfies $x_e = y_e(= 0)$. Furthermore, T^{*+2e} is also a 1–RPP satisfying $x_e = y_e(= 1)$. By substituting them in $ax + by = c$ and by subtracting the corresponding equalities we obtain that $a_e + b_e = 0$.

Let be $f \in E'_{NR} \cup E''_{NR}$, $f \neq e$. Since (V, E_{NR}) is a 3–edge connected graph, $(V, E_{NR} \setminus \{e, f\})$ is a connected graph, and there is a T–join in $(V, E_{NR} \setminus \{e, f\})$ connecting the R–odd vertices. The 1–RPP tour T^{**} build with this T–join does not traverse f nor e and then satisfies $x_e = y_e (= 0)$. Furthermore, T^{**+2f} is also a 1–RPP satisfying $x_e = y_e = 0$. By substituting them in $ax + by = c$ and by subtracting the corresponding equalities, we obtain that $a_f + b_f = 0$. Hence, we have $a_f + b_f = 0 \ \forall f \in E_{NR}$.

For each cycle $\mathcal{C} = \mathcal{C}' \cup \mathcal{C}''$ on $G \setminus \{e\}$ formed by non–required edges, the tour $T^* + \mathcal{C}$ obtained after adding to T^* one copy of each edge in $\mathcal{C}$ (and, then, removing two copies of any edge traversed three times) is also a 1–RPP tour satisfying $x_e = y_e$. Proceeding as in the proof of Theorem 4.2.2 we obtain $b(\mathcal{C}') - b(\mathcal{C}'') = 0$ and, given that $a_f + b_f = 0$, $\forall f \in E_{NR}$, $b(\mathcal{C}') + a(\mathcal{C}'') = 0$ holds.

For each $f = (i, j) \in E_{NR}$, $f \neq e$, since $(V, E_{NR} \setminus \{e\})$ is a 3–edge connected graph, there are two edge–disjoint paths $\mathcal{P}_1$, $\mathcal{P}_2$ joining vertices i and j with non–required edges different from e and f. By considering the three cycles $\mathcal{P}_1 \cup \{f\}$, $\mathcal{P}_2 \cup \{f\}$, and $\mathcal{P}_1 \cup \mathcal{P}_2$, for which equation $b(\mathcal{C}') + a(\mathcal{C}'') = 0$ holds, we obtain $b_f = 0$ if $f \in E'_{NR}$ and $a_f = 0$ if $f \in E''_{NR}$, and, hence, $a_f = b_f = 0$ for all $f \in E_{NR} \setminus \{e\}$. By replacing this and $a_e + b_e = 0$ in (4.25), we obtain $\sum_{f \in E_R} a_f = c$ and we are done. □

Note 4.2.5.1. Inequalities $x_e \geq 0$ and $y_e \leq 1$, for all $e \in E_{NR}$, do not induce a facet of 1–RPP(G) because they are dominated by inequalities $x_e \geq y_e$.

Let us describe now conditions under which connectivity inequalities (4.13) induce a facet of the polyhedron. Note that, given that $x_e = 1$ for all $e \in E_R$, inequalities (4.13) are obviously satisfied if $\delta_R(S) \neq \emptyset$. Hence, we will assume $\delta_R(S) = \emptyset$. Furthermore, if $E(S)$ contains some required edge f, as $x_f = 1$ holds, inequalities (4.13) are dominated by inequalities

$$\sum_{e \in \delta(S)} \big(x_e + y_e\big) \geq 2, \quad \forall S \subseteq V \setminus \{1\} : \ \delta_R(S) = \emptyset, \ E_R(S) \neq \emptyset,$$

which are studied in Theorem 4.2.7.

Theorem 4.2.6. *Let $S \subseteq V \setminus \{1\}$ such that $E_R(S) = \delta_R(S) = \emptyset$. Let $f \in E(S)$ $(f \in E''_{NR})$. The connectivity inequality* (4.13)*, which now takes the form*

$$(x + y)(\delta(S)) \geq 2x_f, \tag{4.26}$$

is facet–inducing for 1–RPP(G) *if subgraph $(S, E_{NR}(S))$ is 3-edge connected and either $V \backslash S = \{1\}$, or subgraph $(V \backslash S, E_{NR}(V \backslash S))$ is 3-edge connected.*

Proof: Let $ax + by \geq c$, i.e.,

$$\sum_{e \in E_R} a_e x_e + \sum_{e \in E_{NR}} a_e x_e + \sum_{e \in E_{NR}} b_e y_e \geq c,$$

be a valid inequality such that

$$\{(x, y) \in 1\text{–RPP}(G) : \ (x+y)(\delta(S)) - 2x_f = 0\} \ \subseteq \ \{(x, y) \in 1\text{–RPP}(G) : \ ax + by = c\}.$$

We have to prove that this inequality is a linear combination of the equalities (4.14) and $(x+y)(\delta(S)) - 2x_f \geq 0$. Note that this means to prove that

$$\begin{aligned}
a_e &= b_e = 0, \quad \forall e \in E_{NR}(S) \setminus \{f\} \cup E_{NR}(V\setminus S),\\
a_e &= b_e = \alpha, \quad \forall e \in \delta(S),\\
a_f &= -2\alpha,\\
b_f &= 0,\\
c &= \sum_{e\in E_R} a_e.
\end{aligned}$$

Consider the 1–RPP tour T' formed with two copies of each edge in $E_{NR}(S) \cup E_{NR}(V\setminus S)$ and then replacing one copy of each $e \in E'_{NR}$ by the required edge parallel to e, plus two copies of a given edge $g \in \delta(S)$. Since this tour satisfies $(x+y)(\delta(S)) - 2x_f = 0$, it also satisfies $ax + by = c$, and we have

$$\sum_{e\in E_R} a_e + \sum_{e\in E_{NR}\setminus\delta(S)} a_e + \sum_{e\in E''_{NR}\setminus\delta(S)} b_e + a_g + b_g = c. \tag{4.27}$$

For each edge $e \in E''_{NR}\setminus\delta(S)$, $e \neq f$, consider the tour T'^{-2e} (it is a 1–RPP tour because subgraphs $G(S)$ and $G(V\setminus S)$ are 2–edge connected). By comparing both tours we obtain $a_e + b_e = 0$ for each edge $e \in E''_{NR}\setminus\delta(S)$, $e \neq f$.

For each edge $e \in E'_{NR}\setminus\delta(S)$, necessarily $e \in E'_{NR}(V\setminus S)$, given that graph $(V\setminus S, E_{NR}(V\setminus S) \setminus \{e\})$ is connected, there is a T–join M connecting its R–odd vertices. The edges in $M \cup E_R$ form an even subgraph in $G(V\setminus S)$, although not necessarily connected. If we add the edges in a closed walk in $E_{NR}(V\setminus S)\setminus\{e\}$ starting at the depot, visiting at least one node in each connected component of $M \cup E_R$ and ending at the depot, we obtain a 1–RPP tour T^* that does not traverse e and satisfies $(x+y)(\delta(S)) = 0 = 2x_f$. The 1–RPP tour T^{*+2e} also satisfies $(x+y)(\delta(S)) = 0$ and, by subtracting the corresponding equations, we obtain that $a_e + b_e = 0$, for each edge $e \in E'_{NR}\setminus\delta(S)$.

For each cycle $\mathcal{C} = \mathcal{C}' \cup \mathcal{C}''$ either on $G(V\setminus S)$ or in $G(S)$ (traversing edge f or not) formed by non–required edges, the tour $T' - \mathcal{C}$ obtained after removing from T' one copy of each edge in $\mathcal{C}$ is also a 1–RPP tour satisfying $(x+y)(\delta(S)) = 2 = 2x_f$ (note that $x_f = 1$ holds). Proceeding as in the proof of Theorem 4.2.2, we obtain $a(\mathcal{C}') + b(\mathcal{C}'') = 0$.

Let $e = (i,j) \in E'_{NR}(V\setminus S)$ (if any). Since that $(V\setminus S, E_{NR}(V\setminus S))$ is a 3–edge connected graph, there are two edge–disjoint paths $\mathcal{P}_1$, $\mathcal{P}_2$ joining vertices i and j with non–required edges different from e. Consider the three cycles $\mathcal{P}_1 \cup \{e\}$, $\mathcal{P}_2 \cup \{e\}$, and $\mathcal{P}_1 \cup \mathcal{P}_2$, for which equation $a(\mathcal{C}') + b(\mathcal{C}'') = 0$ holds. Hence, we have:

$$\begin{aligned}
a(\mathcal{P}'_1) + b(\mathcal{P}''_1) + a_e &= 0,\\
a(\mathcal{P}'_2) + b(\mathcal{P}''_2) + a_e &= 0, \text{ and}\\
a(\mathcal{P}'_1) + b(\mathcal{P}''_1) + a(\mathcal{P}'_2) + b(\mathcal{P}''_2) &= 0,
\end{aligned}$$

and we obtain $a_e = 0$. As $a_e + b_e = 0$ holds, we have $b_e = 0$ for all $e \in E'_{NR}(V\setminus S)$.

Let be $e = (i,j) \in E''_{NR}(V\setminus S)$ (if any). Let $\mathcal{P}_1$, $\mathcal{P}_2$ be two edge–disjoint paths as above and consider the three cycles $\mathcal{P}_1 \cup \{e\}$, $\mathcal{P}_2 \cup \{e\}$, and $\mathcal{P}_1 \cup \mathcal{P}_2$. Now we have:

$$a(\mathcal{P}'_1) + b(\mathcal{P}''_1) + b_e = 0,$$

$$\begin{aligned} a(\mathcal{P}_2') + b(\mathcal{P}_2'') + b_e &= 0, \text{ and} \\ a(\mathcal{P}_1') + b(\mathcal{P}_1'') + a(\mathcal{P}_2') + b(\mathcal{P}_2'') &= 0, \end{aligned}$$

from which we obtain $b_e = 0$ and, as $a_e + b_e = 0$ holds, we have $a_e = 0$ for all $e \in E''_{NR}(V \setminus S)$. Hence, $a_e = b_e = 0$ for all $e \in E_{NR}(V \setminus S)$. In a similar way we obtain $a_e = b_e = 0$ for each edge $e \in E_{NR}(S)$, $e \neq f$, and $b_f = 0$.

Let us denote the edges in $\delta(S)$ as $e_1, \dots, e_p$, where $p \geq 3$ since graph G is 3–edge connected. Now consider two edges $e_1, e_2 \in \delta(S)$. Consider the 1–RPP tour T formed with two copies of each edge in $E_{NR}(S) \cup E_{NR}(V \setminus S)$ and then replacing one copy of each $e \in E'_{NR}$ by the required edge parallel to e, plus two copies of edge e_1. Let T^* be the tour obtained from T after removing the second traversal of e_1 and one copy of each edge of two paths $\mathcal{P}_1, \mathcal{P}_2$ joining the endpoins of e_1 and e_2, and adding the first traversal of e_2. Both tours satisfy $(x+y)(\delta(S)) = 2x_f = 2$ and, after subtracting the corresponding equalities we obtain $a(\mathcal{P}_1') + b(\mathcal{P}_2') + a(\mathcal{P}_1'') + b(\mathcal{P}_2'') + b_{e_1} - a_{e_2} = 0$ and, hence, $b_{e_1} = a_{e_2}$. If we interchange the roles of the edges e_1 and e_2, we obtain $b_{e_2} = a_{e_1}$. Proceeding in this way with all the pairs of edges in $\delta(S)$, we obtain $a_{e_i} = b_{e_j}$ for all $i \neq j \in \{1, \dots, p\}$ and then $a_{e_i} = a_{e_j} = b_{e_i} = b_{e_j}$ for all i, j (because $p \geq 3$ holds).

Let T^* be a 1–RPP tour formed with two copies of each edge in $E_{NR}(V \setminus S)$ and then replacing one copy of each $e \in E'_{NR}$ by the required edge parallel to e, plus two copies of each edge in a given path $\mathcal{P}$ joining a vertex in $G(V \setminus S)$ to an endnode of f traversing $\delta(S)$ once, say, with edge $e \in \delta(S)$, and two copies of edge f. By comparing this tour with the one removing the two copies of the edges in $\mathcal{P}$ and the two copies of f, both satisfying $(x+y)(\delta(S)) = 2x_f$, we obtain that $a_e + b_e + a_f + b_f = 0$. Given that $b_f = 0$ and $a_e = b_e$, we have $a_f = -2a_e$ for any edge $e \in \delta(S)$.

By replacing $a_e = b_e = 0$ for each $e \in E_{NR} \setminus \big(\delta(S) \cup \{f\}\big)$, $a_e = b_e = \alpha$ for each $e \in \delta(S)$, and $b_f = 0$, $a_f = -2\alpha$, in equation (4.27) we obtain $\sum_{e \in E_R} a_e = c$, and after replacing all the previous facts in $ax + by \geq c$ we obtain

$$\sum_{e \in E_R} a_e x_e + \alpha\Big((x+y)(\delta(S)) - 2x_f\Big) \geq \sum_{e \in E_R} a_e,$$

which is a linear combination of the equalities (4.14) and $(x+y)(\delta(S)) \geq 2$. Hence, the connectivity inequality (4.26) is facet–inducing for 1–RPP(G). $\square$

Theorem 4.2.7. *Let $S \subseteq V \setminus \{1\}$ such that $\delta_R(S) = \emptyset$ and $E_R(S) \neq \emptyset$. The connectivity inequality*

$$(x+y)(\delta(S)) \geq 2, \tag{4.28}$$

is facet–inducing for 1–RPP(G) *if subgraph $(S, E_{NR}(S))$ is 3–edge connected and either $V \setminus S = \{1\}$, or subgraph $(V \setminus S, E_{NR}(V \setminus S))$ is 3–edge connected.*

Proof: Let $ax + by \geq c$, i.e.,

$$\sum_{e \in E_R} a_e x_e + \sum_{e \in E_{NR}} a_e x_e + \sum_{e \in E_{NR}} b_e y_e \geq c,$$

be a valid inequality such that

$$\{(x,y) \in 1\text{–RPP}(G) : \quad (x+y)(\delta(S)) = 2\} \subseteq \{(x,y) \in 1\text{–RPP}(G) : \quad ax + by = c\}.$$

We have to prove that this inequality is a linear combination of the equalities (4.14) and $(x+y)(\delta(S)) \geq 2$. Note that this means to prove that

$$\begin{aligned} a_e &= b_e = 0, \quad \forall e \in E_{NR}(S) \cup E_{NR}(V \setminus S), \\ a_e &= b_e = \alpha, \quad \forall e \in \delta(S), \\ c &= \sum_{e \in E_R} a_e + 2\alpha. \end{aligned}$$

Consider the 1–RPP tour T formed with two copies of each edge in $E_{NR}(S) \cup E_{NR}(V \setminus S)$ and then replacing one copy of each $e \in E'_{NR}$ by the required edge parallel to e, plus two copies of a given edge $f \in \delta(S)$. Since this tour satisfies $(x+y)(\delta(S)) = 2$, it also satisfies $ax + by = c$, and we have

$$\sum_{e \in E_R} a_e + \sum_{e \in E_{NR} \setminus \delta(S)} a_e + \sum_{e \in E''_{NR} \setminus \delta(S)} b_e + a_f + b_f = c. \tag{4.29}$$

For each edge $e \in E''_{NR} \setminus \delta(S)$, consider the tour above except for $x_e = y_e = 0$ (it is a 1–RPP tour because subgraphs $G(S)$ and $G(V \setminus S)$ are 2–edge connected). By comparing both tours, we obtain $a_e + b_e = 0$ for each edge $e \in E''_{NR} \setminus \delta(S)$.

For each edge $e \in E'_{NR}(S)$, given that $(S, E_{NR}(S))$ is 2–edge connected, the graph $(S, E_{NR}(S) \setminus \{e\})$ is connected and there is a T–join in it connecting its R–odd vertices that can be completed with edges in $E''_{NR}(S)$ used twice. A similar vector can be defined in the (connected) graph $(V \setminus S, E_{NR}(V \setminus S))$ and, after adding two copies of a given edge $f \in \delta(S)$ we have a 1–RPP tour T^* that does not traverse e and satisfies $(x+y)(\delta(S)) = 2$. The 1–RPP tour T^{*+2e} also satisfies $(x+y)(\delta(S)) = 2$ and, by subtracting the corresponding equations, we obtain that $a_e + b_e = 0$. A similar argument for each edge $e \in E'_{NR}(V \setminus S)$ also leads to $a_e + b_e = 0$. Hence, we have $a_e + b_e = 0$ for each edge $e \in E_{NR} \setminus \delta(S)$.

For each cycle $\mathcal{C} = \mathcal{C}' \cup \mathcal{C}''$ either on $G(V \setminus S)$ or on $G(S)$ formed by non–required edges, the tour $T - \mathcal{C}$ obtained after removing from T one copy of each edge in $\mathcal{C}$ is also a 1–RPP tour satisfying $(x+y)(\delta(S)) = 2$. Proceeding as in the proof of Theorem 4.2.2, we obtain $a(\mathcal{C}') + b(\mathcal{C}'') = 0$.

Let $e = (i,j) \in E'_{NR}(V \setminus S)$ (if any). Since that $(V \setminus S, E_{NR}(V \setminus S))$ is a 3–edge connected graph, there are two edge–disjoint paths $\mathcal{P}_1$, $\mathcal{P}_2$ joining vertices i and j with non–required edges different from e. Consider the three cycles $\mathcal{P}_1 \cup \{e\}$, $\mathcal{P}_2 \cup \{e\}$, and $\mathcal{P}_1 \cup \mathcal{P}_2$, for which equation $a(\mathcal{C}') + b(\mathcal{C}'') = 0$ holds. Hence, we have:

$$\begin{aligned} a(\mathcal{P}'_1) + b(\mathcal{P}''_1) + a_e &= 0, \\ a(\mathcal{P}'_2) + b(\mathcal{P}''_2) + a_e &= 0, \text{ and} \\ a(\mathcal{P}'_1) + b(\mathcal{P}''_1) + a(\mathcal{P}'_2) + b(\mathcal{P}''_2) &= 0, \end{aligned}$$

and we obtain $a_e = 0$. Furthermore, as $a_e + b_e = 0$ holds, we have $b_e = 0$ for all $e \in E'_{NR}(V \setminus S)$.

Let $e = (i,j) \in E''_{NR}(V \setminus S)$ (if any). Let $\mathcal{P}_1$, $\mathcal{P}_2$ be two edge–disjoint paths as above and consider the three cycles $\mathcal{P}_1 \cup \{e\}$, $\mathcal{P}_2 \cup \{e\}$, and $\mathcal{P}_1 \cup \mathcal{P}_2$. Now we have:

$$a(\mathcal{P}'_1) + b(\mathcal{P}''_1) + b_e = 0,$$

$$\begin{aligned} a(\mathcal{P}_2') + b(\mathcal{P}_2'') + b_e &= 0, \text{ and} \\ a(\mathcal{P}_1') + b(\mathcal{P}_1'') + a(\mathcal{P}_2') + b(\mathcal{P}_2'') &= 0, \end{aligned}$$

from which we obtain $b_e = 0$ and, as $a_e + b_e = 0$ holds, we have $a_e = 0$ for all $e \in E''_{NR}(V \setminus S)$. Hence, $a_e = b_e = 0$ for all $e \in E_{NR}(V \setminus S)$. In a similar way, since $(S, E_{NR}(S))$ is a 3–edge connected graph, we obtain $a_e = b_e = 0$ also for each edge $e \in E_{NR}(S)$.

Let us denote the edges in $\delta(S)$ as $e_1, \dots, e_p$, where $p \geq 3$ since graph G is 3–edge connected. The same argument used in Theorem 4.2.6 leads to prove that $a_{e_i} = a_{e_j} = b_{e_i} = b_{e_j}$ for all e_i, e_j.

By replacing $a_e = b_e = 0$, for each $e \in E_{NR} \setminus \delta(S)$, and $a_e = b_e = \alpha$, for each $e \in \delta(S)$, in equation (4.29), we obtain that $\sum_{e \in E_R} a_e + 2\alpha = c$, and after replacing all the previous facts in $ax + by \geq c$, we obtain

$$\sum_{e \in E_R} a_e x_e + \alpha\Big((x+y)(\delta(S))\Big) \geq \sum_{e \in E_R} a_e + 2\alpha,$$

which is a linear combination of the equalities (4.14) and $(x+y)(\delta(S)) \geq 2$. $\square$

In the remaining of the section, we present several new families of valid inequalities for the 1–RPP: parity, $p-$connectivity, and K–C inequalities.

Parity inequalities

In Corberán et al. (2013), the following constraints that generalize the well known co–circuit inequalities (Barahona and Gröetschel, 1986), were proposed for the MBCPP. They are called parity inequalities and, from Theorem 4.2.1, they are also valid for 1–RPP(G):

$$(x-y)(\delta(S) \backslash F) \geq (x-y)(F) - |F| + 1, \quad \forall S \subset V, \ \forall F \subseteq \delta(S) \text{ with } |F| \text{ odd.} \qquad (4.30)$$

The above inequality can be simplified for the 1–RPP taking into account that, for each required edge e, we have $x_e = 1$ and there is no variable y_e. In general, either F and $\delta(S) \backslash F$ could contain required and non–required edges. However, it can be seen that the parity inequalities corresponding to sets where $\delta_R(S) \setminus F \neq \emptyset$ are not facet inducing. Hence, we will assume that $\delta(S) \backslash F \subset E_{NR}$. Let us denote here $F = F_R \cup F_{NR} = F_R \cup F'_{NR} \cup F''_{NR}$. By substituting $x_e = 1$ and deleting variables y_e for $e \in F_R$ in (4.30) we obtain

$$(x-y)(\delta(S) \backslash F) \geq x(F_R) + (x-y)(F_{NR}) - |F| + 1,$$

$$\Downarrow$$

$$(x-y)(\delta(S) \backslash F) \geq |F_R| + (x-y)(F_{NR}) - |F| + 1,$$

$$\Downarrow$$

$$(x-y)(\delta(S) \backslash F) \geq (x-y)(F_{NR}) - |F_{NR}| + 1 \qquad (4.31)$$

This parity inequality (4.31) can be understood in the following way: the 1–RPP tours for which $(x-y)(F_{NR}) = |F_{NR}|$ (all the non–required edges in F traversed once) holds, and given that all the edges in F_R are traversed once and $|F|$ is odd, they satisfy $(x-y)(\delta(S)\backslash F) \geq 1$. For the other 1–RPP tours, the inequality says nothing ($(x-y)(\delta(S)\backslash F) \geq 0$). These inequalities cut off (infeasible) solutions in which there is a cut–set with an odd number of edges traversed exactly once (these edges define the set F) and the other edges are traversed twice or none.

In the case $\delta(S)\backslash F = \emptyset$, and hence $F = \delta(S)$, the parity inequality (4.31) is

$$(x-y)(F_{NR}) \leq |F_{NR}| - 1, \tag{4.32}$$

while in the case $F_{NR} = \emptyset$, the parity inequality (4.31) is

$$(x-y)(\delta(S)\backslash F) \geq 1. \tag{4.33}$$

However, note that both sets, F_{NR} and $\delta(S)\backslash F$, cannot be empty simultaneously, since $F_{NR} \cup \delta(S) \setminus F = \delta_{NR}(S)$ and, as we assume graph (V, E_{NR}) is 3–edge connected, $|\delta_{NR}(S)| \geq 3$ holds.

Note 4.2.7.1. Before proving if some parity inequalities (4.31) induce facets of 1–RPP(G), we will describe two types of 1–RPP tours satisfying them with equality. Recall that $|F|$ is odd. We are going to build 1–RPP tours traversing $\delta(S)$ a number $|F|+1$ or $|F|-1$ of times. Let us consider a cut–set $\delta(S)$ such that graphs $G(S)$ and $G(V \setminus S)$ are connected. We select an even number of (copies of) edges in $\delta(S)$ in the following two ways:

Type 1: $\rightarrow$ If $\delta(S)\backslash F \neq \emptyset$, we select a copy of each edge in F and one more edge in $\delta(S)\backslash F$.

Type 2 $\rightarrow$ If $F_{NR} \neq \emptyset$, we select one copy of each edge in F, except one edge in F_{NR}.

Note that, in both cases, we have selected an even number of copies of edges in $\delta(S)$. Let $V_o \subset V \setminus S$ be the set of vertices incident with an odd number of these selected edges. Given that the number of edges is even, $|V_o|$ is also even, and there is a T–join in $(V \setminus S, E_{NR}(V \setminus S))$. This same process is done in $(S, E_{NR}(S))$. Consider two copies of each non–required edge in $G(V \setminus S)$ and in $G(S)$ not belonging to the T–joins. Now, replace a copy of each edge in E'_{NR} by its corresponding parallel required edge. All these edges plus the two T–joins, plus the selected edges in $\delta(S)$, define a 1–RPP tour (it is even and connected and traverses all the required edges). It satisfies (4.31) with equality because

$$(x-y)(\delta(S)\backslash F) = 1, \quad (x-y)(F_{NR}) = |F_{NR}| \quad (\textit{Type 1}), \quad \text{or}$$

$$(x-y)(\delta(S)\backslash F) = 0, \quad (x-y)(F_{NR}) = |F_{NR}| - 1 \quad (\textit{Type 2}).$$

Theorem 4.2.8. *Parity inequalities* (4.31)*, for all $S \subset V$, $F \subseteq \delta(S)$ with $|F|$ odd and $\delta_R(S) \subseteq F$ (and, hence, $\delta(S)\backslash F \subset E_{NR}$), are facet–inducing for* 1–RPP(G) *if subgraphs $(S, E_{NR}(S))$ and $(V \backslash S, E_{NR}(V \backslash S))$ are 3–edge connected graphs.*

Proof: Let us denote here $F = F_R \cup F_{NR} = F_R \cup F'_{NR} \cup F''_{NR}$. Inequalities (4.31) can be written in the following way:

$$(x-y)(\delta(S)\backslash F) - (x-y)(F_{NR}) \geq 1 - |F_{NR}|. \tag{4.34}$$

Let us suppose there is another valid inequality $ax + by \geq c$,

$$\sum_{e\in E_R} a_e x_e + \sum_{e\in E_{NR}} a_e x_e + e \sum_{e\in E_{NR}} b_e y_e \geq c, \tag{4.35}$$

such that

$$\begin{aligned}\{(x,y)\in 1\text{–RPP}(G):\ (x-y)(\delta(S)\backslash F) - (x-y)(F_{NR}) = 1 - |F_{NR}|\} \subseteq \\ \subseteq \{(x,y)\in 1\text{–RPP}(G):\ \ ax+by=c\}.\end{aligned}$$

We have to prove that inequality (4.35) is a linear combination of equalities (4.14) and $(x-y)(\delta(S)\backslash F) - (x-y)(F_{NR}) \geq 1 - |F_{NR}|$.

Let $e \in E_{NR} \setminus \delta(S)$. Given that graphs $(S, E_{NR}(S))$ and $(V\setminus S, E_{NR}(V\setminus S))$ are 3–edge connected, they remain connected after deleting edge e, and there is a 1–RPP tour T in $G \setminus \{e\}$ that satisfies (4.34) with equality (see Note 4.2.7.1). The 1–RPP tour T^{+2e}, obtained from T by adding two traversals of edge e, also satisfies (4.34) with equality. By comparing the equations $ax + by = c$ corresponding to both tours, we obtain $a_e + b_e = 0$ for all $e \in E_{NR} \setminus \delta(S)$.

Let T be a 1–RPP tour T build as in Note 4.2.7.1 that traverses all the non–required edges in $G(V\setminus S)$ and $G(S)$. If any edge $e' \in E_{NR}$ is not traversed because it was in the T–join and has been replaced by its corresponding parallel edge $e \in E_R$, we add two copies of e' to T. For each cycle $\mathcal{C} = \mathcal{C}' \cup \mathcal{C}''$ either on $G(V\setminus S)$ or in $G(S)$ formed with non–required edges, the tour $T - \mathcal{C}$ obtained after removing from T one copy of each edge in $\mathcal{C}$ is also a 1–RPP tour satisfying (4.34) with equality. Proceeding as in the proof of Theorem 4.2.2, we obtain $a(\mathcal{C}') + b(\mathcal{C}'') = 0$.

Let $e = (i,j) \in E'_{NR}(V\setminus S)$ (if any). Since $(V\setminus S, E_{NR}(V\setminus S))$ is a 3–edge connected graph, there are two edge–disjoint paths $\mathcal{P}_1$, $\mathcal{P}_2$ joining vertices i and j with non–required edges different from e. Consider the three cycles $\mathcal{P}_1 \cup \{e\}$, $\mathcal{P}_2 \cup \{e\}$, and $\mathcal{P}_1 \cup \mathcal{P}_2$, for which equation $a(\mathcal{C}') + b(\mathcal{C}'') = 0$ holds. Hence, we have:

$$\begin{aligned} a(\mathcal{P}'_1) + b(\mathcal{P}''_1) + a_e &= 0, \\ a(\mathcal{P}'_2) + b(\mathcal{P}''_2) + a_e &= 0, \text{ and} \\ a(\mathcal{P}'_1) + b(\mathcal{P}''_1) + a(\mathcal{P}'_2) + b(\mathcal{P}''_2) &= 0, \end{aligned}$$

and we obtain $a_e = 0$. Furthermore, as $a_e + b_e = 0$ holds, we have $a_e = b_e = 0$ for all $e \in E'_{NR}(V\setminus S)$.

Let $e = (i,j) \in E''_{NR}(V\setminus S)$ (if any). Let $\mathcal{P}_1$, $\mathcal{P}_2$ be two edge–disjoint paths as above and consider the three cycles $\mathcal{P}_1 \cup \{e\}$, $\mathcal{P}_2 \cup \{e\}$, and $\mathcal{P}_1 \cup \mathcal{P}_2$. Now we have:

$$\begin{aligned} a(\mathcal{P}'_1) + b(\mathcal{P}''_1) + b_e &= 0, \\ a(\mathcal{P}'_2) + b(\mathcal{P}''_2) + b_e &= 0, \text{ and} \\ a(\mathcal{P}'_1) + b(\mathcal{P}''_1) + a(\mathcal{P}'_2) + b(\mathcal{P}''_2) &= 0, \end{aligned}$$

from which we obtain $b_e = 0$ and, as $a_e + b_e = 0$ holds, we have $a_e = b_e = 0$ for all $e \in E''_{NR}(V \setminus S)$. In a similar way we obtain $a_e = b_e = 0$ for each edge $e \in E_{NR}(S)$. Hence, we have $a_e = b_e = 0$ for all $e \in E_{NR} \setminus \delta(S)$.

Let $e = (i, j) \in \delta_{NR}(S)$. If $e \in F_{NR}$, there is a 1–RPP tour of type 2 not traversing e. If $e \in \delta(S) \setminus F$ and $F_{NR} \neq \emptyset$, then there is a 1–RPP tour of type 2 not traversing e. If $e \in \delta(S) \setminus F$ and $F_{NR} = \emptyset$, then $|\delta(S)\backslash F| \geq 3$ and there is a 1–RPP tour ot type 1 not traversing e (traversing another edge in $\delta(S)\backslash F$). In any case, there is a 1–RPP tour T that satisfies (4.34) with equality and does not traverses e. The tour T^{+2e} also satisfies (4.34) with equality and, by comparing the equations $ax + by = c$ corresponding to both tours, we obtain $a_e + b_e = 0$ for all $e \in \delta_{NR}(S)$.

Let us suppose there are $e_1, e_2 \in F_{NR}$. Let T^1 be the 1–RPP tour of type 2 that traverses once all the edges in F except e_1 (see Note 4.2.7.1), and let T^2 be a similar tour corresponding to edge e_2. Both tours satisfy (4.34) with equality and, by comparing them, and considering that $a_e = b_e = 0$ for all edges $e \in E_{NR} \setminus \delta(S)$, we obtain $a_{e_1} = a_{e_2}$. By iterating this argument, we obtain $a_e = \lambda$ for all $e \in F_{NR}$. Furthermore, since $a_e + b_e = 0$ for each $e \in \delta_{NR}(S)$, we have $b_e = -\lambda$. Hence, $a_e = \lambda$ and $b_e = -\lambda$ for all $e \in F_{NR}$. Note that this is obviously true if F_{NR} contains only one edge.

Let us suppose there are $e_1, e_2 \in \delta(S)\backslash F$. Let T^1 be the 1–RPP tour of type 1 that traverses once the edges in $F \cup \{e_1\}$ (see Note 4.2.7.1), and let T^2 be the tour that traverses once the edges in $F \cup \{e_2\}$. Both tours satisfy (4.34) with equality and, by comparing them, and considering that $a_e = b_e = 0$ for all edges $e \in E_{NR}\backslash\delta(S)$, we obtain $a_{e_1} = a_{e_2}$. By iterating this argument, we obtain $a_e = \mu$ for all $e \in \delta(S)\backslash F$ and, hence, $b_e = -\mu$ for all $e \in \delta(S)\backslash F$. Again, this is obviously true if $\delta(S)\backslash F$ contains only one edge.

If both sets F_{NR} and $\delta(S)\backslash F$ are non–empty, let $e_1 \in F_{NR}$ and $e_2 \in \delta(S)\backslash F$. Let T^1 be the 1–RPP tour of type 1 that traverses once the edges in $F \cup \{e_2\}$, and T^2 the 1–RPP tour of type 2 that does not traverse e_1 nor e_2. Both tours satisfy (4.34) with equality and, by comparing them, we obtain $a_{e_1} + a_{e_2} = 0$ and, therefore, $\lambda = -\mu$. Hence, we have $a_e = \lambda$, $b_e = -\lambda$ for all $e \in F_{NR}$, and $a_e = -\lambda$, $b_e = \lambda$ for all $e \in \delta(S)\backslash F$.

By substituting all the previously computed coefficients a_e, b_e in inequality (4.35), we obtain

$$\sum_{e \in E_R} a_e x_e - \lambda\Big((x-y)(\delta(S)\backslash F) - (x-y)(F_{NR})\Big) \geq c.$$

Given that any of the 1–RPP tours T above satisfies this inequality with equality, we obtain

$$\sum_{e \in E_R} a_e - \lambda\Big(1 - |F_{NR}|\Big) = c$$

and, hence, inequality (4.35) reduces to

$$\sum_{e \in E_R} a_e x_e - \lambda\Big((x-y)(\delta(S)\backslash F) - (x-y)(F_{NR})\Big) \geq \sum_{e \in E_R} a_e - \lambda\Big(1 - |F_{NR}|\Big),$$

which is a linear combination of equalities (4.14) and $(x-y)(\delta(S)\backslash F) - (x-y)(F_{NR}) \geq 1 - |F_{NR}|$. □

Note 4.2.8.1. Theorem 4.2.8 also applies if one of the two shores S or $V \setminus S$ is formed only with one vertex.

p–connectivity inequalities

The constraints we describe in this section were introduced in Corberán et al. (2013) for the MBCPP to cut off fractional solutions as the one described as follows. Consider the 1–RPP instance shown in Figure 4.1a, in which the depot is represented by a black square, each thick line represents a required edge and each thin line represents a non–required one. Consider the fractional solution (x^*, y^*) with values $x^*_{(1,2)} = y^*_{(1,2)} = x^*_{(1,4)} = y^*_{(1,4)} = x^*_{(2,4)} = y^*_{(2,4)} = 0.5$, and $x^*_{(2,3)} = x^*_{(2,3)'} = x^*_{(4,5)} = x^*_{(4,5)'} = 1$, and the remaining variables equal to zero (see Figure 4.1b). It can be seen that this fractional solution satisfies all the parity and connectivity inequalities presented in previous sections, but it is cut off with the p–connectivity inequalities we present in what follows.

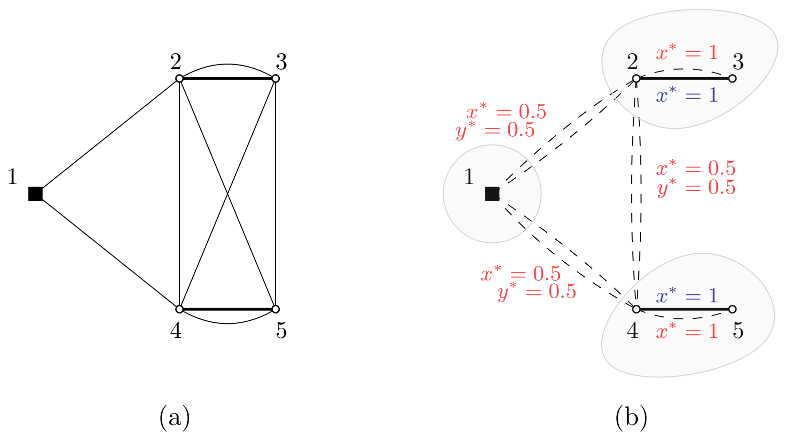


Figure 4.1: A 1–RPP instance and a fractional solution that violates a 2–connectivity inequality

Let $\{S_0, \ldots, S_p\}$ be a partition of V such that $\delta(S_i) \cap E_R = \emptyset$, for each $i = 0, \ldots, p$. Assume we divide the set $\{0, 1, \ldots, p\} = \mathcal{R} \cup \mathcal{N}$ (from 'Required' and 'Non–required') in such a way that

i) $i \in \mathcal{R}$ if either $1 \in S_i$ or $E_R(S_i) \neq \emptyset$ (note that $1 \leq |\mathcal{R}| \leq p+1$), and

ii) $i \in \mathcal{N}$ if $1 \notin S_i$ and $E_R(S_i) = \emptyset$ (note that $0 \leq |\mathcal{N}| \leq p$, and $|\mathcal{R}| + |\mathcal{N}| = p+1$),

and select one edge $e_i \in E(S_i)$ for every $i \in \mathcal{N}$. Note that $e_i \in E''_{NR}$. The following inequality

$$(x+y)(\delta(S_0)) + 2 \sum_{1 \leq r < t \leq p} x(S_r : S_t) \geq 2 \sum_{i \in \mathcal{N}} x_{e_i} + 2\,(|\mathcal{R}| - 1) \tag{4.36}$$

will be referred to as a p–*connectivity inequality*. Note that it is valid for the 1–RPP because the following p–connectivity inequalities

$$(x+y)(\delta(S_0)) + 2 \sum_{1 \leq r < t \leq p} x(S_r : S_t) \geq 2 \sum_{i=0, i \neq d}^{p} x_{e_i} \tag{4.37}$$

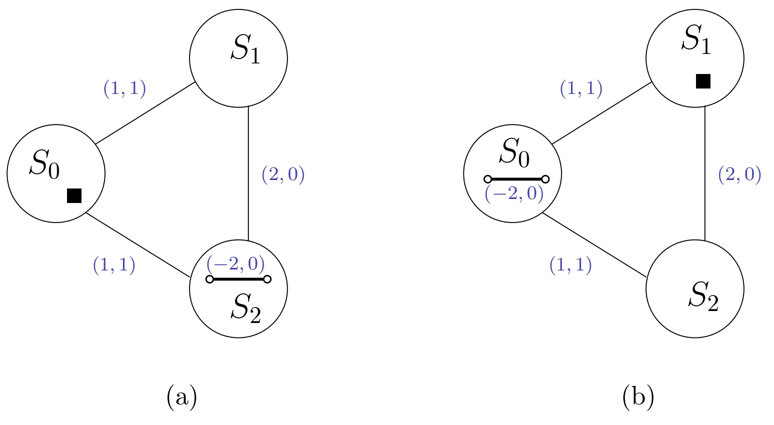


Figure 4.2: Coefficients of a 2–connectivity inequality (4.36)

are valid for the MBCPP (where it is assumed that $1 \in S_d$), and inequalities (4.36) are obtained from them after replacing the equalities $x_{e_i} = 1$ for all $i \in \mathcal{R}$ (required edges).

This inequality with $p = 2$ and $|\mathcal{N}| = 1$ is represented in Figures 4.2(b) and 4.2(c), where for each pair (a, b) associated with an edge e, a and b represent the coefficients of x_e and y_e, respectively. Regarding the fractional solution (x^*, y^*) described above for the instance represented in Figure 4.1, the corresponding p–connectivity inequality (4.36) with $p = 2$ and $1 \in S_0$,

$$x_{(1,2)} + y_{(1,2)} + x_{(1,4)} + y_{(1,4)} + 2(x_{(2,4)} + x_{(2,5)} + x_{(3,4)} + x_{(3,5)}) \geq 4,$$

is violated by (x^*, y^*) (as $1 + 1 + 1 < 4$ holds).

Theorem 4.2.9. *p–connectivity inequalities* (4.36) *are facet–inducing for* 1–RPP(G) *if subgraphs $(S_i, E_{NR}(S_i))$, $\forall i = 0, \dots, p$, are 3–edge connected, $|(S_0 : S_i)| \geq 2$, $\forall i = 1, \dots, p$, and the graph induced by $V \setminus S_0$ is connected.*

Proof: We will assume that $1 \in S_0$. The case $1 \in S_i$, $i \neq 0$, is similar and the proof is omitted here for the sake of brevity. Inequality (4.36) can be written as:

$$(x + y)(\delta(S_0)) + 2 \sum_{1 \leq r < t \leq p} x(S_r : S_t) - 2 \sum_{i \in \mathcal{N}} x_{e_i} \geq 2|\mathcal{R}| - 2. \tag{4.38}$$

Let us suppose there is another valid inequality $ax + by \geq c$,

$$\sum_{e \in E_R} a_e x_e + \sum_{e \in E_{NR}} a_e x_e + \sum_{e \in E_{NR}} b_e y_e \geq c, \tag{4.39}$$

such that

$$\{(x, y) \in 1\text{–RPP}(G) : \; (x + y)(\delta(S_0)) + 2 \sum_{1 \leq r < t \leq p} x(S_r : S_t) - 2 \sum_{i \in \mathcal{N}} x_{e_i} = 2|\mathcal{R}| - 2\} \subseteq$$
$$\subseteq \{(x, y) \in 1\text{–RPP}(G) : \; ax + by = c\}.$$

We have to prove that inequality (4.39) is a linear combination of the equalities (4.14) and inequality (4.38).

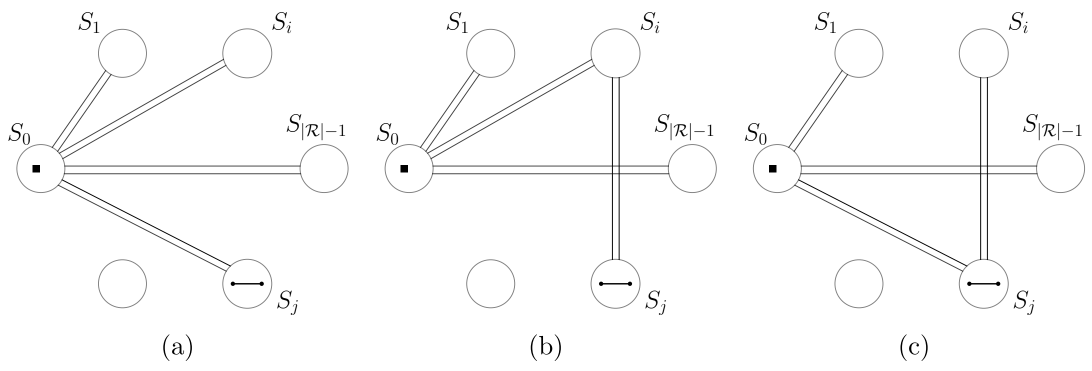


Figure 4.3: 1–RPP tours satisfying (4.38) with equality

In the 1–RPP tours used in this proof we will not describe how the edges in each set $E(S_i)$ are traversed. It can be seen that all these tours can be completed by using T–joins, connecting with non–required edges traversed twice and replacing a traversal of each edge in E'_{NR} by the traversal of its parallel required edge, as described in Note 4.2.7.1 for the parity inequalities.

Similar arguments to those used in the proof of Theorem 4.2.8 lead to prove that $a_e + b_e = 0$, for each $e \in E_{NR}(S_i)$, $i \in \mathcal{R}$ and for each $e \in E_{NR}(S_i) \setminus \{e_i\}$, $i \in \mathcal{N}$. Furthermore, using the 3–edge connectivity of graph $(S_i, E_{NR}(S_i))$ (hence, there are two edge–disjoint paths $\mathcal{P}_1$, $\mathcal{P}_2$ joining the ends of e with non–required edges different from e), we obtain that $b_e = 0$. Hence, we have $a_e = b_e = 0$ for all $e \in E_{NR}(S_i)$, $i \in \mathcal{R}$ and for all $e \in E_{NR}(S_i) \setminus \{e_i\}$, $i \in \mathcal{N}$.

Let S_i and S_j, $i, j \neq 0$ be two sets such that there is an edge $e \in (S_i : S_j)$. Note that $e \in E''_{NR}$. For the sake of simplicity, let us assume $i \in \mathcal{R}$, $j \in \mathcal{N}$ (with the other possibilities we would proceed similarly). Since all the sets $(S_0 : S_k)$ are non–empty, and subgraph $(S_j, E_{NR}(S_j))$ is 3–edge connected, we can construct the 1–RPP tour that traverses twice an edge $f \in (S_0 : S_j)$, traverses once the edge e_j, traverses all the required edges, and visits all the sets S_i, $i \in \mathcal{R}$ (see Figure 4.3a, where we assume $\mathcal{R} = \{0, \dots, |\mathcal{R}| - 1\}$ and $\mathcal{N} = \{|\mathcal{R}|, \dots, p\}$). This tour satisfies inequality (4.38) as an equality. If we compare this tour with the one obtained after removing the two traversals of f and all the traversals of edges in $E(S_j)$, we obtain $a_f + b_f + a_{e_j} = 0$. We construct two more 1–RPP tours satisfying (4.38) with equality such as those depicted in Figure 4.3b and 4.3c. By comparing tours (a) and (b), we obtain $a_{0j} + b_{0j} = a_{ij} + b_{ij} = -a_{e_j}$, and by comparing (a) and (c) we obtain $a_{0i} + b_{0i} = a_{ij} + b_{ij} = -a_{e_j}$, where a_{kl} (b_{kl}) represents the coefficient of the variable x (y) corresponding to any edge in $(S_k : S_l)$. Given that the graph induced by $V \setminus S_0$ is connected, we can iterate this argument to conclude that $a_e + b_e = 2\lambda$ for every edge $e \in (S_i : S_j)$ (including $(S_0 : S_i)$), and $a_{e_i} = -2\lambda$ for each e_i, $i \in \mathcal{N}$. Given that graph $(S_i, E_{NR}(S_i))$ is 3–edge connected and $b_e = 0$ for all edge $e \in E(S_i) \setminus \{e_i\}$, by comparing a 1–RPP tour traversing e_i twice and the tour obtained by replacing the second traversal of e_i by the traversal of a path joining its end–vertices, we obtain $b_{e_i} = 0$ for each e_i, $i \in \mathcal{N}$.

For each $i \in \{1, 2, \dots, p\}$, let e_1, e_2 be two edges in $(S_0 : S_i)$ (recall that $|(S_0 : S_i)| \geq 2$ holds). We have already proved that $a_{e_1} + b_{e_1} = a_{e_2} + b_{e_2} = 2\lambda$. It can be seen that we can construct four 1–RPP tours satisfying inequality (4.38) as an equality as follows. One tour traverses e_1 once and does not traverse e_2. Another tour traverses e_2 once and does not traverse e_1. By comparing these tours, we obtain $a_{e_1} = a_{e_2}$ and, hence, $b_{e_1} = b_{e_2}$. The third tour traverses both e_1 and e_2 once, and the fourth one traverses e_1 twice and does not traverse e_2. By comparing them, we obtain $a_{e_2} = b_{e_1}$ and, hence, also $a_{e_1} = b_{e_2}$, and $a_{e_1} = b_{e_1} = a_{e_2} = b_{e_2} = \lambda$. Hence, $a_e = b_e = \lambda$ for each edge $e \in (S_0 : S_i)$, $i = 1, \dots, p$.

As above, let S_i and S_j, $i, j \neq 0$, be two sets such that there is an edge $e = (u, v) \in (S_i : S_j)$ (again with $i \in \mathcal{R}$, $j \in \mathcal{N}$, for example). There is a 1–RPP tour T that traverses once edge e, an edge $e_i \in (S_0 : S_i)$, and an edge $e_j \in (S_0 : S_j)$, and satisfies inequality (4.38) as an equality. If we remove in T the traversal of e and add the traversal of the edges in a path joining u and v formed with edges e_i, e_j plus some edges in $G(S_0)$, $G(S_i)$ and $G(S_j) \setminus \{e_j\}$ (if any of these last edges is traversed three times, two copies would be removed), we obtain another 1–RPP tour satisfying (4.38) as an equality. By comparing both tours we obtain $a_e = b_{e_i} + b_{e_j} = 2\lambda$, which implies $b_e = 0$ (recall that $a_e + b_e = 2\lambda$). Hence, $a_e = 2\lambda, b_e = 0$, for each edge $e \in (S_i : S_j)$, $i \neq j$.

By substituting all the previously computed coefficients a_e, b_e in inequality (4.39) we obtain

$$\sum_{e \in E_R} a_e x_e - 2\lambda \sum_{i \in \mathcal{N}} x_{e_i} + \lambda x(\delta(S_0)) + 2\lambda \sum_{1 \leq r < t \leq p} x(S_r : S_t) + \lambda y(\delta(S_0)) \geq c,$$

or equivalently,

$$\sum_{e \in E_R} a_e x_e + \lambda (x + y)(\delta(S_0)) + 2\lambda \sum_{1 \leq r < t \leq p} x(S_r : S_t) - 2\lambda \sum_{i \in \mathcal{N}} x_{e_i} \geq c.$$

Given that the 1–RPP tour in Figure 4.3a after removing the two traversals of f and all the traversals in $G(S_j)$, for example, satisfies this inequality with equality, we obtain

$$\sum_{e \in E_R} a_e + 2\lambda \left(|\mathcal{R}| - 1 \right) + 0 + 0 = c,$$

and, hence, inequality (4.39) reduces to

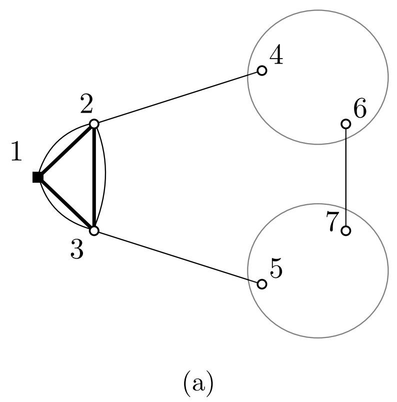
$$\sum_{e \in E_R} a_e x_e + \lambda (x+y)(\delta(S_0)) + 2\lambda \sum_{1 \leq r < t \leq p} x(S_r : S_t) - 2\lambda \sum_{i \in \mathcal{N}} x_{e_i} \geq \sum_{e \in E_R} a_e + \lambda \left(2|\mathcal{R}| - 2 \right),$$

which is a linear combination of the equalities (4.14) and inequality (4.38). □

K–C inequalities

K–C inequalities were introduced and proved to be facet–inducing for the undirected rural postman problem (RPP) in Corberán and Sanchis (1994). We describe here a new version of these inequalities, which we will keep calling K–C inequalities for the sake of simplicity, and prove them to be valid and facet–inducing for the 1–RPP polyhedron.

Let us consider the 1–RPP instance shown in Figure 4.4a, in which the depot is represented by a square labeled 1, each thick line represents a required edge, each thin line represents a non–required one, and each large circle represents an arbitrary



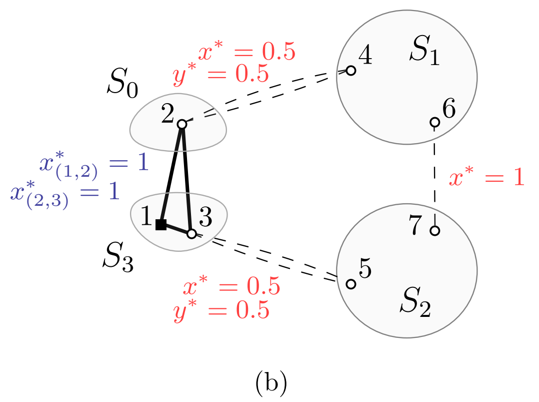


Figure 4.4: A 1–RPP instance and a fractional solution that violates a K–C inequality

subgraph containing at least one required edge. Let (x^*, y^*) be the fractional solution with $x^*_e = 1$ for each required edge e and $x^*_{e'} = y^*_{e'} = 0$ for each of its corresponding parallel non–required edge e', and satisfying $x^*_{(2,4)} = y^*_{(2,4)} = x^*_{(3,5)} = y^*_{(3,5)} = 0.5$ and $x^*_{(6,7)} = 1, y^*_{(6,7)} = 0$ (see Figure 4.4b). This solution is connected but is not even at vertex 2 nor at vertex 3. Furthermore, it cannot be cut off with parity inequalities (4.31). If we consider, for example, the cut–set $\delta(\{2\})$ and $F = \{(1,2),(2,3),(2,4)\}$, we have the associated parity inequality

$$x_{(1,2)'} - y_{(1,2)'} + x_{(2,3)'} - y_{(2,3)'} \geq x_{(2,4)} - y_{(2,4)} - 1 + 1,$$

which is not violated by (x^*, y^*) (as $0 \geq 0$ holds). Note that the fractional solution similar to (x^*, y^*) except for $x^*_{(2,4)} = x^*_{(3,5)} = 1$ and $y^*_{(2,4)} = y^*_{(3,5)} = 0$ is indeed cut off by the above parity inequality. It can also be seen that (x^*, y^*) satisfies all the p–connectivity inequalities (4.36). However, (x^*, y^*) is cut with the inequalities presented in this section.

Let $\{S_0, \dots, S_K\}$, with $K \geq 3$, be a partition of V such that $\delta(S_i) \cap E_R = \emptyset$ for all $i = 1, 2, \dots, K-1$. Assume we divide the set $\{1, \dots, K-1\} = \mathcal{R} \cup \mathcal{N}$ (from 'Required' and 'Non–required') in such a way that

i) $i \in \mathcal{R}$ if either $1 \in S_i$ or $E_R(S_i) \neq \emptyset$ (note that $0 \leq |\mathcal{R}| \leq K-1$), and

ii) $i \in \mathcal{N}$ if $1 \notin S_i$ and $E_R(S_i) = \emptyset$ (note that $0 \leq |\mathcal{N}| \leq K-1$, and $|\mathcal{R}| + |\mathcal{N}| = K-1$),

and select one edge $e_i \in E(S_i)$ for every $i \in \mathcal{N}$. Note that $e_i \in E''_{NR}$. Let $F \subseteq (S_0 : S_K)$ be a set of edges, with $|F| \geq 2$ and even, and $(S_0 : S_K)_R \subseteq F$. Let us denote here $F = F_R \cup F_{NR} = F_R \cup F'_{NR} \cup F''_{NR}$. The K–C inequalities for the 1–RPP are defined as:

$$\begin{aligned}(K-2)(x-y)\Big((S_0 : S_K) \setminus F\Big) - (K-2)(x-y)(F_{NR}) + \\ + \sum_{\substack{0 \leq i < j \leq K \\ (i,j) \neq (0,K)}} \Big((j-i)x(S_i : S_j) + (2-j+i)y(S_i : S_j)\Big) - 2\sum_{i \in \mathcal{N}} x_{e_i} \geq \\ \geq 2|\mathcal{R}| - (K-2)|F_{NR}|. \qquad (4.40)\end{aligned}$$

The coefficients and structure of these K–C inequalities are shown in Figure 4.5, where for the sake of simplicity we assume $\mathcal{R} = \{1, \dots, |\mathcal{R}|\}$ and $\mathcal{N} = \{|\mathcal{R}|+1, \dots, K-$

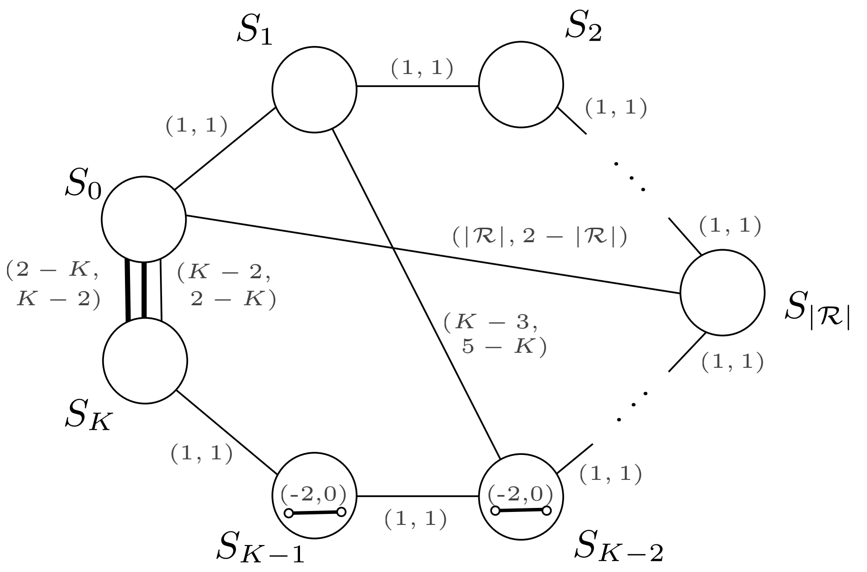


Figure 4.5: Coefficients of a K–C inequality (4.40)

1}. Edges in F (required and non–required, if any) are represented by thick lines. For each pair (a, b) associated with an edge e, a and b represent the coefficients in the inequality of x_e and y_e, respectively. K–C inequalities (4.40) are valid for the 1–RPP(G) because they are obtained from the corresponding K–C inequality for the MBCPP after replacing each x_e by one and removing the y_e variables for all the required edges e. It is easy to see that, when $K = 2$, K–C inequality (4.40) reduces to a connectivity inequality (4.28) when $1 \in \mathcal{R}$, and to a connectivity inequality (4.26) when $1 \in \mathcal{N}$.

Regarding the fractional solution (x^*, y^*) represented in Figure 4.4b, the corresponding K–C inequality (4.40) with $K = 3$, $F = \{(1,2),(2,3)\}$ and $(S_0 : S_3) \setminus F = \{(1,2)', (2,3)'\}$,

$$x_{(1,2)'} - y_{(1,2)'} + x_{(2,3)'} - y_{(2,3)'} + x_{(2,4)} + y_{(2,4)} + x_{(6,7)} + y_{(6,7)} + x_{(3,5)} + y_{(3,5)} \geq 4,$$

is violated by (x^*, y^*) (as $0 + 3 < 4$ holds).

Note 4.2.9.1. Les us describe several types of 1–RPP tours that satisfy the K–C inequality (4.40) with equality that will be used in the proof of Theorem 4.2.10. We do not detail how the edges in each set $E(S_i)$ are traversed. Note that if subgraphs $(S_i, E_{NR}(S_i))$, $i = 0, \ldots, K$, are 3–edge connected, all these tours can be completed by using T–joins as described in Note 4.2.7.1 for the parity inequalities. All of them traverse all the required edges. Note also that, although sets $(S_0 : S_K) \backslash F$ and F_{NR} could be empty sets, they cannot be empty simultaneously, because $F_{NR} \cup (S_0 : S_K) \setminus F = (S_0 : S_K)_{NR}$ and, as in Theorem 4.2.10 is assumed that $|F_R| \geq 2$, $(S_0 : S_K)$ contains at least two non–required edges.

(a) Tours traversing exactly once each edge in F, twice each edge e_i, for all $i \in \mathcal{N}$, and connecting sets S_j, $j = 0, 1, 2, \ldots, K-1$, with either two different edges in $(S_j : S_{j+1})$ used once or an edge used twice, as in Figure 4.6a. Additionally, these tours could also traverse twice any edge (not drawn) in $(S_0 : S_K) \setminus F$. These tours

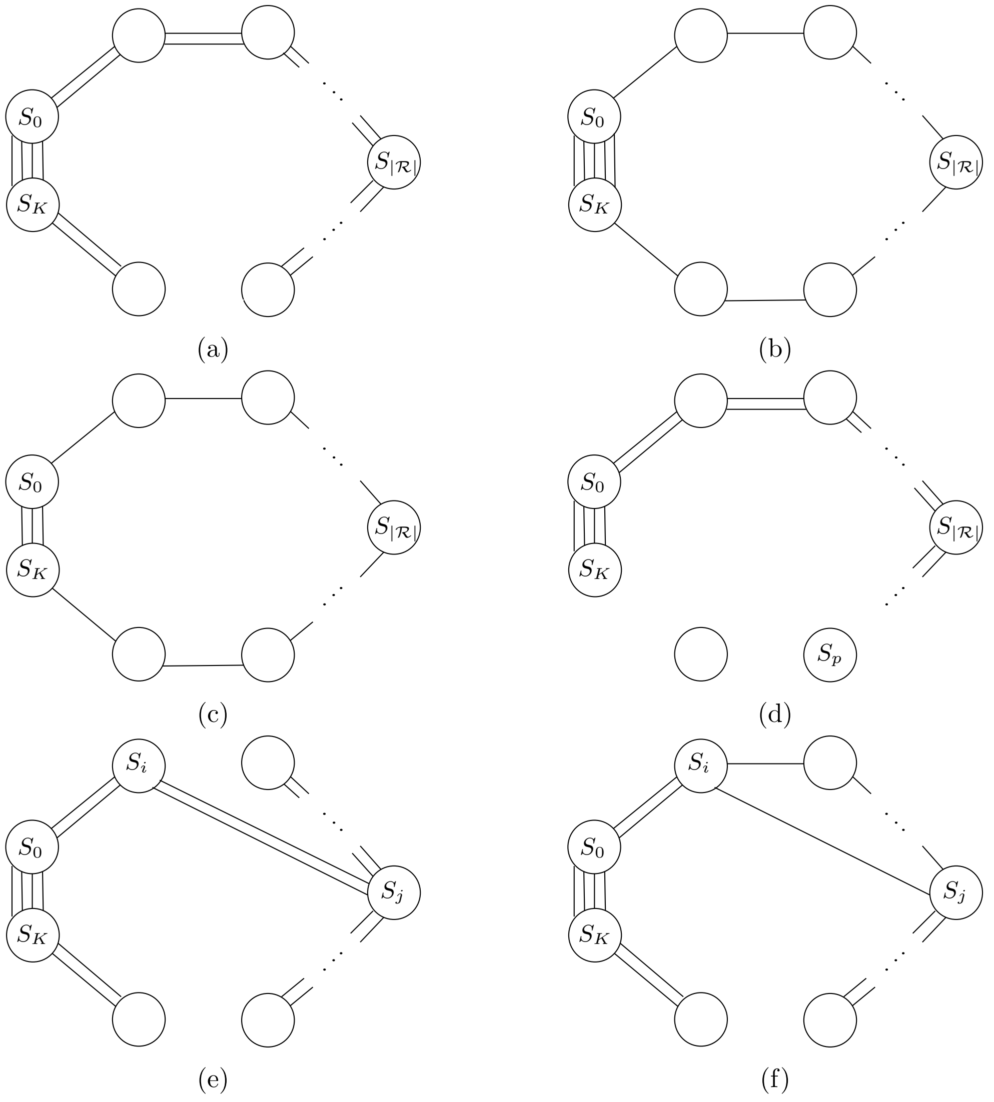


Figure 4.6: 1–RPP tours described in Note 4.2.9.1 and used in the proof of Theorem 4.2.10

satisfy (4.40) with equality:

$$-(K-2)|F_{NR}| + 2(K-1) - 2|\mathcal{N}| = 2|\mathcal{R}| - (K-2)|F_{NR}|.$$

(b) Tours traversing once each edge in F and one more edge in $(S_0 : S_K)$ (this could be a second traversal of an edge in F_{NR}), once each edge e_i, $i \in \mathcal{N}$, and connecting sets S_j, $j = 0, 1, 2, \ldots, K-1$, with exactly an edge in each set $(S_j : S_{j+1}), j = 0, \ldots, K-1$ (see Figure 4.6b). These tours satisfy (4.40) with equality:

$$(K-2) - (K-2)|F_{NR}| + K - 2|\mathcal{N}| = 2|\mathcal{R}| - (K-2)|F_{NR}|.$$

(c) Only if $F_{NR} \neq \emptyset$, tours traversing exactly once each edge in F except one of them in F_{NR}, once each edge e_i, $i \in \mathcal{N}$, and connecting sets S_j, $j = 0, 1, 2, \ldots, K-1$, with exactly an edge in each set $(S_j : S_{j+1}), j = 0, \ldots, K-1$ (see Figure 4.6c).

These tours satisfy (4.40) with equality:

$$-(K-2)\Big(|F_{NR}|-1\Big)+K-2|\mathcal{N}|=2|\mathcal{R}|-(K-2)|F_{NR}|.$$

(d) Tours traversing exactly once each edge in F, twice each edge e_i except one of them, say e_p, and connecting sets $S_j, j \neq p$ as in Figure 4.6d. These tours satisfy (4.40) with equality:

$$-(K-2)|F_{NR}|+2(K-2)-2\Big(|\mathcal{N}|-1\Big)=2|\mathcal{R}|-(K-2)|F_{NR}|.$$

(e) and (f) Tours traversing exactly once each edge in F, twice each edge e_i, for all $i \in \mathcal{N}$, and connecting sets S_j, $j=0,1,2,\ldots,K-1$ as shown in Figure 4.6e and 4.6f. These tours also satisfy (4.40) with equality.

Theorem 4.2.10. *K–C inequalities* (4.40) *are facet–inducing for* 1–RPP(G) *if subgraphs* $(S_i,\ E_{NR}(S_i))$, *for* $i=0,\ldots,K$, *are 3–edge connected,* $|(S_i : S_{i+1})| \geq 2$ *for* $i \leq K-1$, *and* $|F_R| \geq 2$.

Proof: Assume that $1 \in S_0$. The proof for the case $1 \in S_i$, $i \neq 0$, is similar. Let us suppose there is another valid inequality $ax+by \geq c$,

$$\sum_{e\in E_R} a_e x_e + \sum_{e\in E_{NR}} a_e x_e + \sum_{e\in E_{NR}} b_e y_e \geq c, \tag{4.41}$$

such that

$$\{(x,y)\in 1\text{–RPP}(G): \quad (x,y) \text{ satisfies (4.40) with equality}\} \subseteq$$
$$\subseteq \{(x,y)\in 1\text{–RPP}(G): \quad ax+by=c\}.$$

We have to prove that inequality (4.41) is a linear combination of the equalities (4.14) and inequality (4.40).

Let $e \in E_{NR}(S_i)$, $i \in \{0,1,\ldots,K\}$, different from e_i if $i \in \mathcal{N}$. Similar arguments to those used in the proof of Theorem 4.2.8 using the 3–edge connectivity of graph $(S_i, E_{NR}(S_i))$ lead to prove that $a_e = b_e = 0$.

Let $e \in (S_0 : S_K)_{NR}$ and let T be a 1–RPP tour of type (c) in Note 4.2.9.1 that does not traverse edge e. The 1–RPP tour T^{+2e} also satisfies (4.40) with equality, since $x_e = y_e = 1$ and the sum of the coefficients of both variables in (4.40) is zero. By comparing the equations $ax+by=c$ corresponding to both tours, we obtain that $a_e + b_e = 0$, for all $e \in (S_0 : S_K)_{NR}$.

For each $i \in \mathcal{N}$, let T^1 be the 1-RPP tour of type (a) in Note 4.2.9.1 traversing twice an edge in each set $(S_j : S_{j+1})$, $j \neq i$ and let T^2 be the 1–RPP tour of type (d) traversing twice the same edge in each set $(S_j : S_{j+1})$, $j \neq i-1, i$. By comparing the corresponding equations $ax+by=c$ of both tours, we obtain that $a_e + b_e + a_{e_i} + b_{e_i} = 0$ for all $e \in (S_{i-1} : S_i)$. If we consider the 1-RPP tour T^3 of type (a) traversing twice an edge in each set $(S_j : S_{j+1})$, $j \neq i-1$, by comparing the equations corresponding to

T^2 and T^3 we conclude $a_e + b_e + a_{e_i} + b_{e_i} = 0$ for all $e \in (S_i : S_{i+1})$. For each $i \in \mathcal{R}$, let T^1 and T^3 two 1–RPP tours of type (a) defined as above. By comparing them we conclude that $a_e + b_e = a_f + b_f$ for all $e \in (S_{i-1} : S_i)$ and $f \in (S_i : S_{i+1})$. By iterating this argument, we obtain that $a_e + b_e = 2\lambda$ for all $e \in (S_i : S_{i+1})$, $i = 1, \ldots, K-1$, and $a_{e_i} + b_{e_i} = -2\lambda$ for all $i \in \mathcal{N}$, where λ is a certain constant value.

For each $i \in \mathcal{N}$, let T^1 be the 1–RPP tour of type (b) in Note 4.2.9.1 traversing edge $e_i = (u, v)$ once. Given that $(S_i, E_{NR}(S_i))$ is 3–edge connected graph, we can find a path connecting u and v that does not use e_i. If we add this path plus one copy of e_i to T^1, we obtain a 1–RPP tour T^2 also satisfying (4.40) with equality. By comparing both tours, and given that $a_e = b_e = 0$ for all $e \in E(S_i) \setminus \{e_i\}$, we obtain $b_{e_i} = 0$ and, therefore, $a_{e_i} = -2\lambda$.

For each $i \in \{0, 1, 2, \ldots, K-1\}$, let e, f be two edges in $E(S_i : S_{i+1})$ (recall that $|(S_i, S_{i+1})| \geq 2$ holds). There are two 1–RPP tours T^1 and T^2 of type (b) in note 4.2.9.1 traversing edges e and f once, respectively. Comparing both tours, we get $a_e = a_f$. Since we have proved that $a_e + b_e = 2\lambda = a_f + b_f$, we have $b_e = b_f$. Furthermore, let T^3 be a tour of type (a) traversing edge e twice and T^4 a similar tour traversing e and f once. By comparing these tours, we obtain $b_e = a_f$ and, since $a_f = a_e$, we get $a_e = b_e$. Therefore $a_e = b_e = \lambda$ for each edge $e \in E(S_i : S_{i+1})$, for all $i \in \{0, 1, 2, \ldots, K-1\}$.

Let $e \in F_{NR}$ (if any). By comparing the 1–RPP tour of type (b) traversing once all the edges in F except edge e that is traversed twice, and the 1–RPP tour of type (c) traversing once all the edges in F except edge e that is not traversed, we obtain that $a_e + b_e = 0$. On the other hand, by comparing the 1–RPP tour of type (a) traversing once all the edges in F and the previous 1–RPP tour of type (c) we obtain that $a_e + \lambda(K-1) - \lambda = 0$. Hence, $a_e = -\lambda(K-2)$ and $b_e = \lambda(K-2)$.

Let $e \in E(S_0 : S_K) \setminus F$ (if any). By comparing the 1–RPP tour of type (a) traversing once all the edges in F and not traversing e and the same tour by adding two copies of e we obtain that $a_e + b_e = 0$. On the other hand, by comparing the 1–RPP tour of type (a) traversing once all the edges in F and the 1–RPP tour of type (b) traversing once all the edges in $F \cup \{e\}$ we obtain that $-a_e + \lambda(K-1) - \lambda = 0$ and, hence, $a_e = -\lambda(K-2)$.

Finally, for each edge $e \in E(S_i : S_j)$, $|i - j| > 1$, comparing tours of type (a) and (e) in Figure 4.6, we obtain $a_e + b_e = 2\lambda$. Then, comparing tours of type (e) and (f), we obtain $b_e + \lambda\big(|i-j| - 1\big) = \lambda$. Therefore, $b_e = \lambda\big(2 - |i-j|\big)$ and $a_e = \lambda|i-j|$.

By substituting all the previously computed coefficients a_e, b_e in inequality (4.41), we obtain

$$\sum_{e \in E_R} a_e x_e + \lambda(K-2)(x-y)\Big((S_0 : S_K) \setminus F\Big) - \lambda(K-2)(x-y)(F_{NR}) +$$
$$+\lambda \sum_{\substack{0 \leq i < j \leq K \\ (i,j) \neq (0,K)}} \Big((j-i)x(S_i : S_j) + (2-j+i)y(S_i : S_j)\Big) - 2\lambda \sum_{i \in \mathcal{N}} x_{e_i} \geq c.$$

Given that the 1–RPP tour of type (a) in Note 4.2.9.1, for example, satisfies this inequality with equality, we obtain

$$\sum_{e \in E_R} a_e - (K-2)|F_{NR}| + 2\lambda(K-1) - 2\lambda|\mathcal{N}| = c \;\Rightarrow\; \sum_{e \in E_R} a_e + \lambda\Big(2|\mathcal{R}| - (K-2)|F_{NR}|\Big) = c,$$

and, hence, inequality (4.41) is a linear combination of equalities (4.14) and inequality (4.40). □

4.2.2 The K–RPP polyhedron ($K \geq 2$)

Once the 1–RPP polyhedron has been studied, this section focuses on the study of the K–RPP, with $K \geq 2$, on graph $G = (V, E) = (V_R, E_R \cup E'_{NR} \cup E''_{NR})$. Recall that K–RPP(G) denotes the polytope defined as the convex hull of the vectors $(x^1, y^1, x^2, y^2, \dots, x^K, y^K) \in \mathbb{Z}^{(2|E_{NR}|+|E_R|)K}$ corresponding to K–RPP solutions.

Theorem 4.2.11. *If (V, E_{NR}) is a 3–edge connected graph, then K–RPP(G) is a full–dimensional polyhedron, i.e., dim(K–RPP(G))= $K(2|E_{NR}| + |E_R|)$.*

Proof: Consider the 1–RPP defined on G and its associated polytope 1–RPP(G). From Theorem 4.2.2, and since (V, E_{NR}) is 3–edge connected, we know that dim(1–RPP(G))= $2|E_{NR}| = m$. Since $0 \notin$ aff(1–RPP(G)), because all its points satisfy equations (4.14), there exist $m + 1$ affinely and linearly independent 1–RPP tours $z_1, z_2, \dots, z_{m+1}$ on G, each $z_i = (x_i, y_i)$ satisfying $x_i(e) = 1$ for all the edges in E_R. We can assume that one of them, say $z_1 = (x_1, y_1)$ is formed with two copies of each edge in E_{NR} and then replacing one copy of each $e \in E'_{NR}$ by the required edge parallel to e. Hence, it satisfies $x_1(e) = 1$ for all $e \in E_R$, $x_1(e) = 1$, $y_1(e) = 0$ for all $e \in E'_{NR}$, and $x_1(e) = y_1(e) = 1$ for all $e \in E''_{NR}$. We can build $(m+1)K$ K–RPP solutions in the following way. One drone performs any 1–RPP tour z_j above, while the other drones perform z_1. These $(m+1)K$ solutions are depicted as the rows of the three first block rows in the matrix shown in Figure 4.7, where, for the sake of simplicity, we have supposed we have $K = 3$ drones.

Furthermore, we can build $K|E_R|$ more K–RPP solutions in the following way. Consider the 1–RPP solution z_1 above. For each required edge $e \in E_R$, we define the vector $t_{(e)} = (x_{(e)}, y_{(e)})$ equal to z_1 except for the entries $x_{(e)}(e) = x_{(e)}(e') = 0$, where e' denotes the non–required edge parallel to e. This vector represents a tour on graph G because its support graph is even and connected. For any required edge e, one drone performs $t_{(e)}$ while the other drones perform z_1. These $K|E_R|$ solutions are depicted as the rows of the three last block rows in the matrix shown in Figure 4.7, where the required edges are $\{e_1, e_2, \dots, e_{|E_R|}\}$.

If we subtract the first row from all the other rows and then we remove the null rows, we obtain the matrix in Figure 4.8, where all the non–depicted values are zero, and a big zero in a block means that all the entries of this block are zero. Its $K(2|E_{NR}|+|E_R|)$ rows are linearly independent because the first -1 entry in each pair $-1, -1$ in the rows of the three last block rows is associated with each edge $e \in E_R$, and since $z_i - z_1$ takes value zero in all the required edges, it is the only non–zero entry in the column corresponding to e. Hence, we have $K(2|E_{NR}| + |E_R|) + 1$ affinely independent K–RPP solutions, and we are done. □

In the following, we will assume that (V, E_{NR}) is a 3–edge connected graph and thus K–RPP(G) is full–dimensional. Therefore, every facet of the polyhedron is induced by a unique inequality (except scalar multiples). As in Theorem 4.2.11 above, in the proofs

$$\begin{array}{ccc}
\text{Drone 1} & \text{Drone 2} & \text{Drone 3} \\
\end{array}$$
$$\left(\begin{array}{c:c:c}
z_1 & z_1 & z_1 \\
z_2 & z_1 & z_1 \\
\cdots & \cdots & \cdots \\
z_{m+1} & z_1 & z_1 \\
\hdashline
z_1 & z_1 & z_1 \\
z_1 & z_2 & z_1 \\
\cdots & \cdots & \cdots \\
z_1 & z_{m+1} & z_1 \\
\hdashline
z_1 & z_1 & z_1 \\
z_1 & z_1 & z_2 \\
\cdots & \cdots & \cdots \\
z_1 & z_1 & z_{m+1} \\
\hline
t_{(e_1)} & z_1 & z_1 \\
\cdots & \cdots & \cdots \\
t_{(e_{|E_R|})} & z_1 & z_1 \\
\hdashline
z_1 & t_{(e_1)} & z_1 \\
\cdots & \cdots & \cdots \\
z_1 & t_{(e_{|E_R|})} & z_1 \\
\hdashline
z_1 & z_1 & t_{(e_1)} \\
\cdots & \cdots & \cdots \\
z_1 & z_1 & t_{(e_{|E_R|})}
\end{array}\right)$$

Figure 4.7: Matrix of K–RPP solutions appearing in the proof of Theorem 4.2.11

of the theorems in this section we will represent the K–RPP solutions that we define only for $K=3$ drones, although they can be extended to any value of K.

Note 4.2.11.1. The vectors $z_1, z_2, \ldots, z_{m+1}$, from the dimension of 1–RPP(G), and $t_{(e)}$, for each $e \in E_R$, defined in the proof of Theorem 4.2.11, will be used also in the proofs of the following theorems. In particular, $z_1 = (\bar{x}_1, \bar{y}_1)$ satisfies $\bar{x}_1(e) = 1$ for all $e \in E_R$, $\bar{x}_1(e) = 1$, $\bar{y}_1(e) = 0$ for all $e \in E'_{NR}$, and $\bar{x}_1(e) = \bar{y}_1(e) = 1$ for all $e \in E''_{NR}$. For each $e \in E_R$, we define $t_{(e)} = (x_{(e)}, y_{(e)})$ equal to z_1 except for the entries $x_{(e)}(e) = x_{(e)}(e') = 0$, where e' denotes the non–required edge parallel to e.

Facet–inducing inequalities obtained from the formulation

We will study here some conditions under which some trivial inequalities and inequalities (4.2), (4.4), and (4.5) from the formulation induce facets of K–RPP(G).

Theorem 4.2.12. *Inequality $y_e^k \geq 0$, for each edge $e \in E_{NR}$, and for each drone $k \in \{1, 2, \ldots, K\}$, is facet–inducing of K–RPP(G).*

Proof: Let us suppose first that $e \in E'_{NR}$. Consider the 1–RPP defined on G and its associated polytope 1–RPP(G). Given that $y_e \geq 0$ is facet–inducing of 1–RPP(G) (Theorem 4.2.3), there exist $m = 2|E_{NR}|$ affinely independent 1–RPP tours $w_1, w_2, \ldots, w_m$

$$
\begin{array}{c}
\begin{array}{ccc}
\text{Drone 1} & \text{Drone 2} & \text{Drone 3}
\end{array}\\
\left(\begin{array}{c:c:c}
\begin{matrix} z_2 - z_1 \\ \cdots \\ z_{m+1} - z_1 \end{matrix} & 0 & 0 \\
\hdashline
0 & \begin{matrix} z_2 - z_1 \\ \cdots \\ z_{m+1} - z_1 \end{matrix} & 0 \\
\hdashline
0 & 0 & \begin{matrix} z_2 - z_1 \\ \cdots \\ z_{m+1} - z_1 \end{matrix} \\
\hline
\begin{matrix} -1-1 & & \\ & \ddots & \\ & & -1-1 \end{matrix} & 0 & 0 \\
\hdashline
0 & \begin{matrix} -1-1 & & \\ & \ddots & \\ & & -1-1 \end{matrix} & 0 \\
\hdashline
0 & 0 & \begin{matrix} -1-1 & & \\ & \ddots & \\ & & -1-1 \end{matrix}
\end{array}\right)
\end{array}
$$

Figure 4.8: Matrix appearing in the proof of Theorem 4.2.11

on G, each $w_i = (x_i, y_i)$, satisfying $x_i(a) = 1$ for all $a \in E_R$, and $y_i(e) = 0$. Consider also the tours $z_1, z_2, \ldots, z_{m+1}$ from Note 4.2.11.1 and assume that $z_1 = w_1$, since $z_1 = (\bar{x}_1, \bar{y}_1)$ satisfies also $\bar{y}_1(e) = 0$.

We can build K–RPP solutions in the following way. Drone k performs any 1–RPP tour w_j above, while the other drones perform z_1. These m solutions are depicted as the rows of the first block row in the matrix shown in Figure 4.9, where we assume that drone k is the first one. Now, a drone different from k performs any tour z_j above, while the other drones do z_1. These $(m+1)(K-1)$ solutions are depicted as the rows of the second and third block rows in the matrix shown in Figure 4.9.

Furthermore, we can build $K|E_R|$ more K–RPP solutions as in the proof of Theorem 4.2.11 with the same vectors $t_{(e_j)}$ for each required edge in $E_R = \{e_1, e_2, \ldots, e_{|E_R|}\}$. Note that the corresponding vectors $t_{(e_j)}$ also satisfy $y_{(e_j)}(e) = 0$. These solutions are depicted in the three last block rows in the matrix in Figure 4.9. As in Theorem 4.2.11, if we subtract the first row from all the other rows and then we remove the null rows, we obtain a matrix similar to that in Figure 4.8 but with $K(2|E_{NR}| + |E_R|) - 1$ rows LI. Hence, we have $K(2|E_{NR}| + |E_R|)$ solutions affinely independent satisfying $y_e^k = 0$, and we are done.

Let us suppose now that $e \in E''_{NR}$. Given that $y_e \geq 0$ is facet–inducing of 1–RPP(G) (Theorem 4.2.3), there exist $m = 2|E_{NR}|$ affinely independent 1–RPP tours $w_1, w_2, \ldots, w_m$ on G, each $w_i = (x_i, y_i)$, satisfying $x_i(a) = 1$, for all $a \in E_R$, and $y_i(e) = 0$. Since dim(1–RPP(G))= $2|E_{NR}| = m$, there exist $m+1$ 1–RPP tours $z_1, z_2, \ldots, z_{m+1}$, each $z_i = (\bar{x}_i, \bar{y}_i)$ satisfying $\bar{x}_i(a) = 1$ for all $a \in E_R$. We can assume

$$
\begin{array}{ccc}
\text{Drone 1} & \text{Drone 2} & \text{Drone 3} \\
\end{array}
$$
$$
\left(\begin{array}{c:c:c}
z_1 = w_1 & z_1 & z_1 \\
w_2 & z_1 & z_1 \\
\dots & \dots & \dots \\
w_m & z_1 & z_1 \\
\hdashline
z_1 & z_1 & z_1 \\
z_1 & z_2 & z_1 \\
\dots & \dots & \dots \\
z_1 & z_{m+1} & z_1 \\
\hdashline
z_1 & z_1 & z_1 \\
z_1 & z_1 & z_2 \\
\dots & \dots & \dots \\
z_1 & z_1 & z_{m+1} \\
\hdashline
t_{(e_1)} & z_1 & z_1 \\
\dots & \dots & \dots \\
t_{(e_{|E_R|})} & z_1 & z_1 \\
\hdashline
z_1 & t_{(e_1)} & z_1 \\
\dots & \dots & \dots \\
z_1 & t_{(e_{|E_R|})} & z_1 \\
\hdashline
z_1 & z_1 & t_{(e_1)} \\
\dots & \dots & \dots \\
z_1 & z_1 & t_{(e_{|E_R|})}
\end{array}\right)
$$

Figure 4.9: Matrix appearing in the proof of Theorem 4.2.12

here that one of them, say z_1, satisfies $\bar{x}_1(a) = 1$ for all $a \in E_R$, $\bar{x}_1(a) = 1$, $\bar{y}_1(a) = 0$ for all $a \in E'_{NR}$, $\bar{x}_1(a) = \bar{y}_1(a) = 1$ for all $a \in E''_{NR}$, $a \neq e$, and $\bar{x}_1(e) = \bar{y}_1(e) = 0$. Thereby, we can assume that $z_1 = w_1$. Furthermore, for each required edge $a \in E_R$, the corresponding vector $t_{(a)}$ is equal to z_1 except for the entries $x_{(a)}(a) = x_{(a)}(a') = 0$, and also satisfies $y_{(a)}(e) = 0$. Hence, we can build the K–RPP solutions as in the matrix in Figure 4.9 and the remainder of the proof is similar to the previous case. $\square$

Theorem 4.2.13. *Inequality $x_e^k \leq 1$, for each edge $e \in E_{NR}$ and for each drone $k \in \{1, 2, \dots, K\}$, is facet–inducing of K–RPP(G).*

Proof: Let us suppose first that edge e is in E''_{NR}. Given that $x_e \leq 1$ is facet–inducing of 1–RPP(G) (Theorem 4.2.4), there exist $m = 2|E_{NR}|$ affinely independent 1–RPP tours $w_1, w_2, \dots, w_m$ on G, each $w_i = (x_i, y_i)$, satisfying $x_i(a) = 1$ for all $a \in E_R$, and $x_i(e) = 1$. Furthermore, the 1–RPP tours $z_1, z_2, \dots, z_{m+1}$ from Note 4.2.11.1 also satisfy $\bar{x}_i(e) = 1$, and we can assume that $z_1 = w_1$. In addition, the vectors $t_{(e_j)}$ defined as in Note 4.2.11.1 for each required edge in $E_R = \{e_1, e_2, \dots, e_{|E_R|}\}$ also satisfy $x_{(e_j)}(e) = 1$. Hence, we can build K–RPP solutions satisfying $x_e^k = 1$ similar to those in the matrix shown in Figure 4.9, where we assume that drone k is the first one, and the remaining of the proof is similar to that of Theorem 4.2.12.

Let us suppose now that the edge is in E'_{NR} and for simplicity is represented by e', while its parallel required edge is represented by e. Again, given that $x_{e'} \leq 1$ is facet–

$$
\begin{array}{ccc}
\text{Drone 1} & \text{Drone 2} & \text{Drone 3} \\
\end{array}
$$
$$
\left(\begin{array}{c:c:c}
z_1 = w_1 & z_1 & z_1 \\
w_2 & z_1 & z_1 \\
\cdots & \cdots & \cdots \\
w_m & z_1 & z_1 \\
\hdashline
z_1 & z_1 & z_1 \\
z_1 & z_2 & z_1 \\
\cdots & \cdots & \cdots \\
z_1 & z_{m+1} & z_1 \\
\hdashline
z_1 & z_1 & z_1 \\
z_1 & z_1 & z_2 \\
\cdots & \cdots & \cdots \\
z_1 & z_1 & z_{m+1} \\
\hline
t_{(e_2)} & z_1 & z_1 \\
\cdots & \cdots & \cdots \\
t_{(e_{|E_R|})} & z_1 & z_1 \\
\hdashline
z_1 & t_{(e_1)} & z_1 \\
\cdots & \cdots & \cdots \\
z_1 & t_{(e_{|E_R|})} & z_1 \\
\hdashline
z_1 & z_1 & t_{(e_1)} \\
\cdots & \cdots & \cdots \\
z_1 & z_1 & t_{(e_{|E_R|})} \\
\hline
t^* & z_1 & z_1
\end{array}\right)
$$

Figure 4.10: Matrix appearing in the proof of Theorem 4.2.13 (case $e \in E'_{NR}$)

inducing of 1–RPP(G) (Theorem 4.2.4), there exist $m = 2|E_{NR}|$ affinely independent 1–RPP tours $w_1, w_2, \ldots, w_m$ on G, each $w_i = (x_i, y_i)$, satisfying $x_i(a) = 1$ for all $a \in E_R$, and $x_i(e') = 1$. The 1–RPP tours $z_1, z_2, \ldots, z_{m+1}$ from Note 4.2.11.1 also satisfy $\bar{x}_i(e') = 1$, and we can assume that $z_1 = w_1$. Hence, we can build the $(m+1)(K-1)$ solutions depicted as the rows of the following block rows in the matrix shown in Figure 4.10, where we assume that drone k is the first one.

In addition, the vectors $t_{(a)}$ defined as in Note 4.2.11.1 for each required edge $a \in E_R$ satisfy $x_{(a)}(e) = 1$ except the one corresponding to edge e. Hence, we can build the $K|E_R| - 1$ K–RPP solutions depicted as the rows of the following block rows in the matrix shown in Figure 4.10, where the required edges are $E_R = \{e_1, e_2, \ldots, e_{|E_R|}\}$ with $e_1 = e$.

Finally, consider the vector $t^* = (x^*, y^*)$ equal to z_1 except for the entries $x^*(e) = 0$ and $y^*(e') = 1$. Note that in t^* we have replaced in z_1 the traversal of edges e and e' by two traversals of e'. Hence, if drone k performs t^* and the other drones perform z_1, we have a K–RPP solution in which the edge e is traversed by a drone different from k (last row in Figure 4.10).

By subtracting the first row from all the other rows and then removing the null rows, we obtain the matrix in Figure 4.11 with $K(2|E_{NR}| + |E_R|) - 1$ rows linearly independent, where an '$*$' represents any value. Note that the first entry (-1) of the last

$$\begin{array}{c}
\begin{array}{ccc} \quad\text{Drone 1}\quad & \quad\text{Drone 2}\quad & \quad\text{Drone 3}\quad \end{array}\\
\left(\begin{array}{c|c|c}
\begin{matrix} w_2-w_1\\ \cdots \\ w_m-w_1\end{matrix} & 0 & 0 \\ \hline
0 & \begin{matrix} z_2-z_1\\ \cdots\\ z_{m+1}-z_1\end{matrix} & 0\\ \hline
0 & 0 & \begin{matrix} z_2-z_1\\ \cdots\\ z_{m+1}-z_1\end{matrix}\\ \hline
\begin{matrix}-1\,-1 & & \\ & \ddots & \\ & & -1\,-1\end{matrix} & 0 & 0\\ \hline
0 & \begin{matrix}-1\,-1 & & \\ & \ddots & \\ & & -1\,-1\end{matrix} & 0\\ \hline
0 & 0 & \begin{matrix}-1\,-1 & & \\ & \ddots & \\ & & -1\,-1\end{matrix}\\ \hline
-1\;0\;0\quad \ast\quad\quad \ast\quad \ast\quad \ast & 0 & 0
\end{array}\right)
\end{array}$$

Figure 4.11: Matrix appearing in the proof of Theorem 4.2.13 (case $e \in E'_{NR}$)

row, associated with edge e, is the only non zero value in its column. Hence, we have $K(2|E_{NR}|+|E_R|)$ solutions affinely independent satisfying $x^k_{e'} = 1$, and we are done. □

Theorem 4.2.14. *Inequality* (4.5) $x^k_e \geq y^k_e$, *for each edge* $e \in E_{NR}$ *and for each drone* k, *is facet–inducing of* K–RPP(G) *if graph* $(V, E_{NR} \setminus \{e\})$ *is 3–edge connected.*

Proof: Let us suppose first that $e \in E''_{NR}$. Given that $x_e \geq y_e$ is facet–inducing of 1–RPP(G) (Theorem 4.2.5), there exist $m = 2|E_{NR}|$ affinely independent 1–RPP tours $w_1, w_2, \ldots, w_m$ on G, each $w_i = (x_i, y_i)$, satisfying $x_i(a) = 1$ for all $a \in E_R$, and $x_i(e) = y_i(e)$. Furthermore, the 1–RPP tours $z_1, z_2, \ldots, z_{m+1}$ from Note 4.2.11.1 also satisfy $\bar{x}_i(e) = \bar{y}_i(e) (= 1)$, and we can assume that $z_1 = w_1$. In addition, the vectors $t_{(e_j)}$ defined as in Note 4.2.11.1 for each edge $e_j \in E_R$ also satisfy $x_{(e_j)}(e) = y_{(e_j)}(e)$. Hence, we can build K–RPP solutions similar to those in the matrix shown in Figure 4.9, all of them satisfying $x^k_e = y^k_e$. The remaining of the proof is similar to that of Theorem 4.2.12.

Let us suppose now that the edge is in E'_{NR} and for simplicity is represented by e', while its parallel required edge is represented by e. Again, given that $x_{e'} \geq y_{e'}$ is facet–inducing of 1–RPP(G) (Theorem 4.2.5), there exist $m = 2|E_{NR}|$ affinely independent 1–RPP tours $w_1, w_2, \ldots, w_m$ on G, each $w_i = (x_i, y_i)$, satisfying $x_i(a) = 1$ for all $a \in E_R$, and $x_i(e') = y_i(e')$. The 1–RPP tours $z_1, z_2, \ldots, z_{m+1}$ from Note 4.2.11.1 satisfy $\bar{x}_i(e) = \bar{x}_i(e') = 1$ and $\bar{y}_i(e') = 0$. We can assume that $w_1 = z_1$ except on the three

$$
\begin{array}{ccc}
\text{Drone 1} & \text{Drone 2} & \text{Drone 3} \\
\end{array}
$$
$$
\left(
\begin{array}{c:c:c}
w_1 & z_1 & z_1 \\
w_2 & z_1 & z_1 \\
\cdots & \cdots & \cdots \\
w_m & z_1 & z_1 \\
\hdashline
w_1 & z_1 & z_1 \\
w_1 & z_2 & z_1 \\
\cdots & \cdots & \cdots \\
w_1 & z_{m+1} & z_1 \\
\hdashline
w_1 & z_1 & z_1 \\
w_1 & z_1 & z_2 \\
\cdots & \cdots & \cdots \\
w_1 & z_1 & z_{m+1} \\
\hdashline
t^*_{(e_2)} & z_1 & z_1 \\
\cdots & \cdots & \cdots \\
t^*_{(e_{|E_R|})} & z_1 & z_1 \\
\hdashline
w_1 & t_{(e_1)} & z_1 \\
\cdots & \cdots & \cdots \\
w_1 & t_{(e_{|E_R|})} & z_1 \\
\hdashline
w_1 & z_1 & t_{(e_1)} \\
\cdots & \cdots & \cdots \\
w_1 & z_1 & t_{(e_{|E_R|})} \\
\hdashline
t^* & z_1 & z_1
\end{array}
\right)
$$

Figure 4.12: Matrix appearing in the proof of Theorem 4.2.14 (case $e \in E'_{NR}$)

entries corresponding to e and e'. Hence, we can build the $(m+1)(K-1)$ solutions depicted as the rows of the first three block rows in the matrix shown in Figure 4.12, where we assume that drone k is the first one.

In addition to the vectors $t_{(a)}$ defined from z_1 in Note 4.2.11.1 for each edge $a \in E_R$, for each edge $a \in E_R$, $a \neq e$, we define $t^*_{(a)}$ equal to w_1 after removing from it the traversals of a and a'. These tours $t^*_{(a)}$ satisfy $x_{(a')}(e) = y_{(a')}(e)$. Hence, we can build the $K|E_R|-1$ K–RPP solutions depicted as the rows of the following block rows in the matrix shown in Figure 4.12.

A last K–RPP tour t^* is obtained from w_1 by removing the edge e and adding all the edges in a path joining its endpoints and, then, deleting two copies of each edge in the path traversed three times, if any. If drone k performs t^* and the other drones perform z_1, we have a K–RPP solution (last row in Figure 4.12).

By subtracting the first row from all the other rows and then removing the null rows, we would obtain a matrix similar to that in Figure 4.11 except in the first entry of the last row that is 1 in this case. The $K(2|E_{NR}|+|E_R|)-1$ rows are linearly independent and, hence, we have $K(2|E_{NR}|+|E_R|)$ solutions affinely independent satisfying $x^k_{e'} = 1$, and we are done. □

Theorem 4.2.15. *Inequality $x^k_e \leq 1$, for each edge $e \in E_R$ and for each drone $k \in$*

*$\{1,\dots,K\}$, is facet–inducing of K–*RPP*(G).*

Proof: All the K–RPP solutions shown in Figure 4.7, except the first one of the fourth row block (the one with drone k performing $t_{(e_1)}$), are affinely independent K–RPP solutions satisfying $x_e^k = 1$. □

Theorem 4.2.16. *Inequality $x_e^k \geq 0$, for each edge $e \in E_R$ and for each drone $k \in \{1,\dots,K\}$, is facet–inducing of K–*RPP*(G).*

Proof: Consider the graph G^* obtained by deleting the required edge e from G. Note that the corresponding non–required parallel edge e', which in graph G belongs to E'_{NR}, is in the set E''_{NR} in graph G^*. Hence, G and G^* have the same set of non–required edges and, from Theorem 4.2.2, dim(1–RPP(G^*))= $2|E_{NR}| = m$ and there exist $m+1$ affinely and linearly independent 1–RPP tours $z_1^*, z_2^*, \dots, z_{m+1}^*$ on G^*, all of them traversing the required edges $a \in E_R \setminus \{e\}$. Each z_i^* is transformed into a 1–RPP tour w_i on G by adding to it an extra entry x_e corresponding to edge e, with value $x_e = 0$. Each $w_i = (x_i, y_i)$, satisfies $x_e = 0$ and $x_i(a) = 1$ for all $a \in E_R \setminus \{e\}$. We can assume that one of them, say w_1, satisfies $x_1(a) = 1$ for all $a \in E_R \setminus \{e\}$, $x_1(a') = 1$, $y_1(a') = 0$ for all $a' \in E'_{NR} \setminus \{e'\}$ and $x_1(a) = y_1(a) = 1$ for all $a \in E''_{NR} \cup \{e'\}$. Consider also the 1–RPP tours $z_1, z_2, \dots, z_{m+1}$ on G from Note 4.2.11.1, each $z_i = (\bar{x}_i, \bar{y}_i)$, satisfying $\bar{x}_i(a) = 1$ for all $a \in E_R$. We can build $(m+1)K$ K–RPP solutions as those depicted in the rows of the three block rows in the matrix shown in Figure 4.13, where we assume that drone k is the first one.

In addition to the vectors $t_{(a)}$ defined from z_1 in Note 4.2.11.1 for each edge $a \in E_R$, for each edge $a \in E_R$, $a \neq e$, we define $t^*_{(a)}$ equal to w_1 after removing from it the traversals of a and a'. These tours $t^*_{(a)}$ satisfy $x_{(a)}(e) = 0$. Hence, we can build the $K|E_R| - 1$ K–RPP solutions depicted as the rows of the last block rows in the matrix shown in Figure 4.13.

If $K \geq 3$, all the rows in the matrix in Figure 4.13 represent K–RPP solutions satisfying $x_e^k = 0$. If we subtract the first row from all the other rows and then we remove the null rows, we would obtain a matrix, with similar structure to that in Figure 4.8, with $K(2|E_{NR}|+|E_R|)-1$ linearly independent rows. Hence, we have $K(2|E_{NR}|+|E_R|)$ affinely independent solutions, and we are done.

If $K = 2$, the solution corresponding to the first row of the last block row of the corresponding matrix, $[w_1, t_{(e_1)}]$, is not actually a K–RPP solution. Note that neither drone 1, performing w_1, nor drone 2, performing $t_{(e_1)}$, traverses this edge. In this case, we consider the solution in which drone 1 performs $t^*_{(e_2)}$ and drone 2 performs $t_{(e_1)}$, which is a K–RPP solution. It can be seen that the corresponding matrix is also full rank. □

Theorem 4.2.17. *Inequalities $\sum_{k=1}^{K} x_e^k \geq 1$, for each $e \in E_R$, are facet–inducing of K–*RPP*(G).*

Proof: Consider the 1–RPP tours $z_1, z_2, \dots, z_{m+1}$ and $t_{(a)}$, $a \in E_R$, from Note 4.2.11.1.

$$\begin{array}{ccc}
\text{Drone 1} & \text{Drone 2} & \text{Drone 3}
\end{array}$$
$$\left(\begin{array}{c:c:c}
w_1 & z_1 & z_1 \\
w_2 & z_1 & z_1 \\
\cdots & \cdots & \cdots \\
w_{m+1} & z_1 & z_1 \\
\hdashline
w_1 & z_1 & z_1 \\
w_1 & z_2 & z_1 \\
\cdots & \cdots & \cdots \\
w_1 & z_{m+1} & z_1 \\
\hdashline
w_1 & z_1 & z_1 \\
w_1 & z_1 & z_2 \\
\cdots & \cdots & \cdots \\
w_1 & z_1 & z_{m+1} \\
\hdashline
t^*_{(e_2)} & z_1 & z_1 \\
\cdots & \cdots & \cdots \\
t^*_{(e_{|E_R|})} & z_1 & z_1 \\
\hdashline
w_1 & t_{(e_1)} & z_1 \\
\cdots & \cdots & \cdots \\
w_1 & t_{(e_{|E_R|})} & z_1 \\
\hdashline
w_1 & z_1 & t_{(e_1)} \\
\cdots & \cdots & \cdots \\
w_1 & z_1 & t_{(e_{|E_R|})}
\end{array}\right)$$

Figure 4.13: Matrix appearing in the proof of Theorem 4.2.16

Let w_1 the tour on G obtained by replacing in z_1 the traversal of edge e by a second traversal of the corresponding parallel edge e'.

We can build $(m+1)K$ K–RPP solutions in the following way. A drone performs any tour z_j above, while the other drones perform w_1. These $(m+1)K$ solutions are depicted as the rows of the three block rows in the matrix shown in Figure 4.14. Now, for any required edge $a \neq e$, a drone performs $t_{(a)}$ while the other drones perform w_1 (see Figure 4.14, where the required edges different from e are $\{e_2, \dots, e_{|E_R|}\}$). All the rows in the matrix depicted in Figure 4.14 represent K–RPP solutions satisfying $\sum_{k=1}^{K} x_e^k = 1$.

If we subtract the first row from all the other rows we obtain the matrix in Figure 4.15. This matrix is shown in more detail in Figure 4.16, where the three leftmost entries in each block k correspond to the variables $x_e^k, x_{e'}^k, y_{e'}^k$, and vectors v_i represent the remaining vectors $z_i - z_1$ (and vectors $z_i - w_1$), $i = 2, \dots, m+1$. Block (1,1) has full rank. If in blocks (2,2) and (3,3) we subtract the first row from the remaining rows, we would obtain two blocks where rows 2 to $m+1$ are identical to those in block (1,1). Hence, the rows in the first three block rows are linearly independent. Furthermore, regarding the last three block rows of the matrix, the value -1 corresponding to each required edge $a \neq e$ is the only non–zero value in its column. Hence, any of these rows can be obtained as a linear combination of the other rows. Therefore, all the $m + (K-1)(m+1) + K(|E_R| - 1) = K(2|E_{NR}| + |E_R|) - 1$ rows of the matrix in Figure 4.16 are linearly independent and, hence, we have $K(2|E_{NR}| + |E_R|)$ K–RPP solutions

$$
\begin{array}{c}
\begin{array}{ccc}
\text{Drone 1} & \text{Drone 2} & \text{Drone 3}
\end{array}\\
\left(\begin{array}{c:c:c}
z_1 & w_1 & w_1 \\
\cdots & \cdots & \cdots \\
z_{m+1} & w_1 & w_1 \\
\hdashline
w_1 & z_1 & w_1 \\
\cdots & \cdots & \cdots \\
w_1 & z_{m+1} & w_1 \\
\hdashline
w_1 & w_1 & z_1 \\
\cdots & \cdots & \cdots \\
w_1 & w_1 & z_{m+1} \\
\hdashline
t_{(e_2)} & w_1 & w_1 \\
\cdots & \cdots & \cdots \\
t_{(e_{|E_R|})} & w_1 & w_1 \\
\hdashline
w_1 & t_{(e_2)} & w_1 \\
\cdots & \cdots & \cdots \\
w_1 & t_{(e_{|E_R|})} & w_1 \\
\hdashline
w_1 & w_1 & t_{(e_2)} \\
\cdots & \cdots & \cdots \\
w_1 & w_1 & t_{(e_{|E_R|})}
\end{array}\right)
\end{array}
$$

Figure 4.14: Matrix appearing in the proof of Theorem 4.2.17

affinely independent satisfying $\sum_{k=1}^{K} x_e^k = 1$, and the proof is complete. $\square$

Theorem 4.2.18. *Connectivity inequalities* (4.2), $x^k\big(\delta_R(S)\big) + (x^k + y^k)\big(\delta_{NR}(S)\big) \geq 2x_f^k$, $\forall S \subseteq V\backslash\{1\}$, $\forall f \in E(S)$, *and* $\forall k$, *are facet–inducing of* K–RPP(G) *if subgraph* $(S, E_{NR}(S))$ *is 3–edge connected and either* $|V\backslash S| = 1$ *or subgraph* $(V\backslash S, E_{NR}(V\backslash S))$ *is 3–edge connected.*

Proof: In some points of this proof we will distinguish the cases $f \in E_{NR}$ and $f \in E_R$. Let $z_1, z_2, \ldots, z_{m+1}$ and $t_{(a)}$, $a \in E_R$, be the 1–RPP tours from Note 4.2.11.1. Consider the graph G^* obtained by deleting from G the required edges in $\delta_R(S) \cup E_R(S)$. Graphs G and G^* have the same set of non–required edges and, given that the hypotheses of Theorem 4.2.6 are fulfilled, there are m affinely and linearly independent 1–RPP tours $z_1^*, z_2^*, \ldots, z_m^*$ on G^*, all of them traversing the required edges $a \in E_R(V \setminus S)$ and satisfying $(x+y)(\delta(S)) = 2x_{\bar{f}}$, where, if edge $f \in E_R$, $\bar{f} = f'$, and $\bar{f} = f$ otherwise. Each z_j^* is transformed into a tour on G, w_j, by adding to it an extra entry x_e with value $x_e = 0$ for each removed edge $e \in \delta_R(S) \cup E_R(S)$. In the case $f \in E_R$, we replace a traversal of f' (if any) by the traversal of f. It can be seen that the resulting tours $w_1, w_2, \ldots, w_m$ on G are also affinely independent, satisfy $x(\delta_R(S)) + (x+y)(\delta_{NR}(S)) = 2x_f$, and do not traverse any edge in $\delta_R(S) \cup E_R(S)$ (except f if it is required). We can assume that one of them, say w_1, is equal to z_1 in the entries corresponding to $E(V \setminus S)$, it traverses the cutset $\delta(S)$ twice through a non–required edge, say $\bar{e}$, and $x_1(a) = y_1(a) = 1$ for all $a \in E_{NR}(S)$, $x_1(a) = 0$ for all $a \in E_R(S)$, in the case $f \in E_{NR}$, and $x_1(f) = x_1(f') = 1$, $y_1(f') = 0$, and $x_1(a) = 0$ for all $a \in E_R(S) \setminus \{f\}$, in the case $f \in E_R$.

As in previous theorems, we build the K–RPP solutions depicted in the first three

$$\begin{array}{ccc}
\text{Drone 1} & \text{Drone 2} & \text{Drone 3} \\
\left(\begin{array}{c:c:c}
z_2 - z_1 & & \\
\cdots & 0 & 0 \\
z_{m+1} - z_1 & & \\ \hdashline
w_1 - z_1 & z_1 - w_1 & \\
\cdots & \cdots & 0 \\
w_1 - z_1 & z_{m+1} - w_1 & \\ \hdashline
w_1 - z_1 & & z_1 - w_1 \\
\cdots & 0 & \cdots \\
w_1 - z_1 & & z_{m+1} - w_1 \\ \hdashline
t_{(e_2)} - z_1 & & \\
\cdots & 0 & 0 \\
t_{(e_{|E_R|})} - z_1 & & \\ \hdashline
w_1 - z_1 & t_{(e_2)} - w_1 & \\
\cdots & \cdots & 0 \\
w_1 - z_1 & t_{(e_{|E_R|})} - w_1 & \\ \hdashline
w_1 - z_1 & & t_{(e_2)} - w_1 \\
\cdots & 0 & \cdots \\
w_1 - z_1 & & t_{(e_{|E_R|})} - w_1
\end{array}\right) & &
\end{array}$$

Figure 4.15: Matrix appearing in the proof of Theorem 4.2.17

block rows in the matrix shown in Figure 4.17 (again is assumed $k = 1$) . We build also the solutions depicted in the last two block rows in the matrix shown in Figure 4.17, where a drone different from k performs any $t_{(a)}$, while drone k performs w_1 and the remaining drones perform z_1.

For each required edge $a \in E_R(V \setminus S)$, we define the vector $t^*_{(a)} = (x^*_{(a)}, y^*_{(a)})$ equal to w_1 except for the entries $x^*_{(a)}(a) = x^*_{(a)}(a') = 0$. For each required edge $a \in \delta_R(S)$, we define the vector $t^*_{(a)} = (x^*_{(a)}, y^*_{(a)})$ equal to w_1 except for the entries $x^*_{(a)}(\bar{e}) = y^*_{(a)}(\bar{e}) = 0$ and $x^*_{(a)}(a) = x^*_{(a)}(a') = 1$. Note that if $\bar{e} = a'$, then $t^*_{(a)}$ is equal to w_1 except for the entries $y^*_{(a)}(a') = 0$ and $x^*_{(a)}(a) = 1$.

In the case $f \in E_{NR}$, for each required edge $a \in E_R(S)$, we define the vector $t^*_{(a)} = (x^*_{(a)}, y^*_{(a)})$ equal to w_1 except for the entries $x^*_{(a)}(a) = 1$ and $y^*_{(a)}(a') = 0$ (we replace the second traversal of a' by the traversal of a). In the case $f \in E_R$, for each $a \in E_R(S) \setminus \{f\}$, we define the vectors $t^*_{(a)} = (x^*_{(a)}, y^*_{(a)})$ as before. Moreover, we define a new vector $t^*_{(f)}$ equal to w_1 in $E(V \setminus S)$ and zero in all the remaining entries. Note that this vector also satisfies $x(\delta_R(S)) + (x+y)(\delta_{NR}(S)) = 2x_f = 0$. If drone k performs $t^*_{(a)}$ and the remaining drones perform z_1 we obtain the K–RPP solutions depicted as the rows of the fourth block rows in the matrix shown in Figure 4.17.

If $K \geq 3$ (the case $K = 2$ is argued at the end of this proof), all the rows in the matrix depicted in Figure 4.17 represent K–RPP solutions satisfying the connectivity inequality (4.2) as an equality. If we subtract the first row from all the other rows and then remove the zero rows we obtain the matrix in Figure 4.18. It can be seen that the rows of block B^*, associated with vectors $t^*_{(a)} - w_1$, have the three entries corresponding

$$\left(\begin{array}{c:c:c}
\begin{matrix}0\ \alpha\ \beta & v_2\\ \cdots & \cdots \\ 0\ \lambda\ \mu & v_{m+1}\end{matrix} & 0 & 0 \\ \hdashline
\begin{matrix}-1\ 0\ 1\\ -1\ 0\ 1\\ \cdots\\ -1\ 0\ 1\end{matrix}\qquad 0 & \begin{matrix}1\ 0 & -1 & 0\\ 1\ \alpha & (\beta-1) & v_2\\ & \cdots & \cdots \\ 1\ \lambda & (\mu-1) & v_{m+1}\end{matrix} & 0 \\ \hdashline
\begin{matrix}-1\ 0\ 1\\ -1\ 0\ 1\\ \cdots\\ -1\ 0\ 1\end{matrix}\qquad 0 & 0 & \begin{matrix}1\ 0 & -1 & 0\\ 1\ \alpha & (\beta-1) & v_2\\ & \cdots & \cdots \\ 1\ \lambda & (\mu-1) & v_{m+1}\end{matrix} \\ \hdashline
\begin{matrix}0\ *\ * & -1\,-1 & \\ \cdots & \ddots & \\ 0\ *\ * & & -1\,-1\end{matrix} & 0 & 0 \\ \hdashline
\begin{matrix}-1\ 0\ 1\\ \cdots\\ -1\ 0\ 1\end{matrix}\qquad 0 & \begin{matrix}1\ *\ * & -1\,-1 & \\ \cdots & \ddots & \\ 1\ *\ * & & -1\,-1\end{matrix} & 0 \\ \hdashline
\begin{matrix}-1\ 0\ 1\\ \cdots\\ -1\ 0\ 1\end{matrix}\qquad 0 & 0 & \begin{matrix}1\ *\ * & -1\,-1 & \\ \cdots & \ddots & \\ 1\ *\ * & & -1\,-1\end{matrix}
\end{array}\right)$$

Figure 4.16: Matrix appearing in the proof of Theorem 4.2.17

to each $a \in E_R$ and its corresponding parallel edge a' as follows: $x(a) = x(a') = -1$, $y(a') = 0$, for each $a \in E_R(V \setminus S)$, $x(a) = x(a') = 1$, $y(a') = 0$, for each $a \in \delta_R(S)$, and $x(a) = 1$, $x(a') = 0$, $y(a') = -1$, for each $a \in E_R(S)$ in the case $f \in E_{NR}$, while in the case $f \in E_R$, $x(a) = 1$, $x(a') = 0$, $y(a') = -1$, for each $a \in E_R(S) \setminus \{f\}$, and $x(f) = -1$, $x(f') = -1$, $y(f') = 0$.

In the case $f \in E_{NR}$, the matrix obtained after merging blocks (1,1) and B^* of Figure 4.18 is detailed in Figure 4.19, with the columns and rows rearranged. Note that, for $a \in E_R$ the corresponding values -1 or 1 are the only non zero entries in its corresponding column of the matrix in Figure 4.18. In the case $f \in E_R$, the entry corresponding to edge f in the block (4,4) of this matrix, is -1 instead of 1. In both cases, this matrix has full rank. Hence, the matrix of Figure 4.18 has full rank and its $m-1+(K-1)m+K|E_R| = K(2|E_{NR}|+|E_R|)-1$ rows are linearly independent. Hence, we have $K(2|E_{NR}| + |E_R|)$ K–RPP solutions affinely independent satisfying inequality (4.2) as an equality, and we are done.

If $K = 2$, the solutions corresponding to some rows of the last block row of the corresponding matrix, $[w_1, t_{(a_i)}]$, are not actually K–RPP solutions. Note that, if $a_i \in \delta_R(S) \cup E_R(S)$, neither drone 1, performing w_1, nor drone 2, performing $t_{(a_i)}$, traverses this edge. In this case, for each $a_i \in \delta_R(S) \cup E_R(S)$, we consider the solution in which drone 1 performs $t^*_{(a_i)}$ (similar to w_1 but traversing a_i) and drone 2 performs $t_{(a_i)}$, which is a K–RPP solution. It can be seen that the corresponding matrix is also full rank. □

$$
\begin{array}{c}
\begin{array}{ccc}
\text{Drone 1} & \text{Drone 2} & \text{Drone 3}
\end{array}\\
\left(\begin{array}{c:c:c}
w_1 & z_1 & z_1 \\
w_2 & z_1 & z_1 \\
\dots & \dots & \dots \\
w_m & z_1 & z_1 \\
\hdashline
w_1 & z_1 & z_1 \\
w_1 & z_2 & z_1 \\
\dots & \dots & \dots \\
w_1 & z_{m+1} & z_1 \\
\hdashline
w_1 & z_1 & z_1 \\
w_1 & z_1 & z_2 \\
\dots & \dots & \dots \\
w_1 & z_1 & z_{m+1} \\
\hdashline
t^*_{(a_1)} & z_1 & z_1 \\
\dots & \dots & \dots \\
t^*_{(a_{|E_R|})} & z_1 & z_1 \\
\hdashline
w_1 & t_{(a_1)} & z_1 \\
\dots & \dots & \dots \\
w_1 & t_{(a_{|E_R|})} & z_1 \\
\hdashline
w_1 & z_1 & t_{(a_1)} \\
\dots & \dots & \dots \\
w_1 & z_1 & t_{(a_{|E_R|})}
\end{array}\right)
\end{array}
$$

Figure 4.17: Matrix appearing in the proof of Theorem 4.2.18

$$
\begin{array}{c}
\begin{array}{cccc}
E_{NR} & E_R(V\backslash S) & \delta_R(S) & E_R(S)
\end{array}\\
\left(\begin{array}{c:c:c:c}
\begin{matrix}(w_2-w_1)_{NR}\\ \dots \\ (w_m-w_1)_{NR}\end{matrix} & 0 & 0 & 0 \\
\hdashline
* & \begin{matrix}-1 & & \\ & \ddots & \\ & & -1\end{matrix} & 0 & 0 \\
\hdashline
* & 0 & \begin{matrix}1 & & \\ & \ddots & \\ & & 1\end{matrix} & 0 \\
\hdashline
* & 0 & 0 & \begin{matrix}1 & & \\ & \ddots & \\ & & 1\end{matrix}
\end{array}\right)
\end{array}
$$

Figure 4.19: A submatrix of the matrix in Figure 4.18

In what follows, we will present the parity, p–connectivity and K–C inequalities for the K–RPP, and study conditions under which they are facet–defining of K–RPP(G).

$$
\begin{array}{ccc}
\text{Drone 1} & \text{Drone 2} & \text{Drone 3} \\
\left(\begin{array}{c:c:c}
\begin{array}{c} w_2 - w_1 \\ \cdots \\ w_m - w_1 \end{array} & 0 & 0 \\
\hdashline
0 & \begin{array}{c} z_2 - z_1 \\ \cdots \\ z_{m+1} - z_1 \end{array} & 0 \\
\hdashline
0 & 0 & \begin{array}{c} z_2 - z_1 \\ \cdots \\ z_{m+1} - z_1 \end{array} \\
\hdashline
B^* & 0 & 0 \\
\hdashline
0 & \begin{array}{ccc} -1 -1 & & \\ & \ddots & \\ & & -1 -1 \end{array} & 0 \\
\hdashline
0 & 0 & \begin{array}{ccc} -1 -1 & & \\ & \ddots & \\ & & -1 -1 \end{array}
\end{array}\right)
\end{array}
$$

Figure 4.18: Matrix appearing in the proof of Theorem 4.2.18

Parity inequalities

Since parity inequalities (4.30) are valid for the MBCPP on G, from Theorem 4.2.1 we have that the following parity inequalities

$$x^k(\delta_R(S)\backslash F) + (x^k - y^k)(\delta_{NR}(S)\backslash F) \geq x^k(F_R) + (x^k - y^k)(F_{NR}) - |F| + 1, \qquad (4.42)$$

for each drone k, any $S \subset V$, and $F \subseteq \delta(S)$ with $|F|$ odd, are valid for the K–RPP on G.

Theorem 4.2.19. *Parity inequalities* (4.42) *are facet–inducing of K–*RPP(G) *if subgraph* (S, $E_{NR}(S)$) *is 3–edge connected and either* $V\backslash S = \{1\}$, *or subgraph* $(V\backslash S, E_{NR}(V\backslash S))$ *is 3–edge connected.*

Proof: Let $z_1, z_2, \ldots, z_{m+1}$ and $t_{(a)}$, $a \in E_R$, the 1–RPP tours from Note 4.2.11.1. Consider the graph G^* obtained by deleting from G the required edges in $\delta_R(S)\backslash F$ (if any). Graphs G and G^* have the same set of non–required edges and, given that the hypotheses of Theorem 4.2.8 are fulfilled, there are m affinely and linearly independent 1–RPP tours $w_1^*, w_2^*, \ldots, w_m^*$ on G^*, all of them traversing the required edges in G^* (all the edges in E_R except the removed ones, in $\delta_R(S) \backslash F$, if any), and satisfying inequalities (4.31) as equalities. These tours also satisfy $x_e = 1$ for all $e \in F_R$ and, by adding $x(F_R) - |F_R| = 0$ to the right hand side of (4.31) we obtain that the tours w_j^* satisfy

$$(x - y)(\delta_{NR}(S)\backslash F) = x(F_R) + (x - y)(F_{NR}) - |F| + 1.$$

$$\begin{array}{c:c:c:c}
E_{NR} & \multicolumn{2}{c}{E_R\setminus(\delta_R(S)\setminus F_R)} & \delta_R(S)\setminus F_R \\
\begin{matrix}(w_2-w_1)_{NR}\\ \cdots \\ (w_m-w_1)_{NR}\end{matrix} & 0 & 0 & 0 \\
\hdashline
* & \begin{matrix}-1 & & \\ & \ddots & \\ & & -1\end{matrix} & 0 & 0 \\
\hdashline
* & 0 & \begin{matrix}-1 & & \\ & \ddots & \\ & & -1\end{matrix} & 0 \\
\hdashline
* & 0 & 0 & \begin{matrix}1 & & \\ & \ddots & \\ & & 1\end{matrix}
\end{array}$$

Figure 4.20: Matrix appearing in the proof of Theorem 4.2.19

Each w_j^* is transformed into a tour on G, w_j, by adding to it an extra entry x_e for each removed edge $\delta_R(S)\setminus F$ with value $x_e = 0$. The resulting tours $w_1, w_2, \dots, w_m$ on G are also affinely independent, traverse all the required edges except those in $\delta_R(S)\setminus F$, and satisfy

$$x(\delta_R(S)\setminus F) + (x-y)(\delta_{NR}(S)\setminus F) = x(F_R) + (x-y)(F_{NR}) - |F| + 1.$$

Regarding the traversal of the cut–set $\delta(S)$, there are only two types of 1–RPP tours w_i that satisfy the inequality as an equality (see Note 4.2.7.1). Tours of type 1 traverse one copy of each edge in F and one more edge in $\delta(S)\setminus F$ (if $\delta(S)\setminus F \neq \emptyset$). Tours of type 2 traverse one copy of each edge in F, except one edge in F_{NR} (if $F_{NR} \neq \emptyset$). We can assume, for example, that $F_{NR} \neq \emptyset$ and that one of the w_i above, say $w_1 = (x_1, y_1)$, traverses once each edge in F except a given edge $e \in F_{NR}$ (in the case $F_{NR} = \emptyset$, and hence, $\delta(S)\setminus F \neq \emptyset$, we would proceed in a similar way).

We build the K–RPP solutions depicted in the first three block rows in the matrix shown in Figure 4.17, where vectors z_i are the same but vectors w_i are different but denoted equal. We build also the solutions depicted in the last two block rows in the matrix in Figure 4.17.

For each required edge a we define a vector $t^*_{(a)}$, obtained from w_1, that also satisfies the inequality as an equality (it is of one of the types 1 or 2 above) and, regarding the required edges, only differs from w_1 in the traversals of edge a in the following way (if any edge in E_{NR} turns to be traversed three times, we would delete two traversals of it):

- For each $a \in E_R(S)$, the vector $t^*_{(a)}$ is obtained from w_1 by replacing the traversal of a with the traversal of a path joining its two endpoints formed with edges in $E_{NR}(S)$. For each $a \in E_R(V\setminus S)$ we proceed in the same way.
- For each $a \in F_R$, the vector $t^*_{(a)}$ is obtained from w_1 by replacing the traversal of a with the traversal of edge e and two paths, formed with edges in $E_{NR}(S)$ and

$E_{NR}(V \setminus S)$, joining the endpoints of a and e.

- For each $a \in \delta_R(S) \setminus F$, the vector $t^*_{(a)}$ is obtained from w_1 by adding the traversal of a cycle formed with edges a and e and two paths with edges in $E_{NR}(S)$ and in $E_{NR}(V \setminus S)$.

If drone k performs $t^*_{(a)}$ and the remaining drones perform z_1 we obtain the K–RPP solutions depicted as the rows of the fourth block rows in the matrix shown in Figure 4.17, although note that vectors $t^*_{(a)}$ are different but share the same name.

If $K \geq 3$ (for the case $K = 2$, the argument is similar to that in the proof of Theorem 4.2.18), all the rows in the matrix depicted in Figure 4.17 represent K–RPP solutions satisfying the parity inequality (4.42) as an equality. If we subtract the first row from all the other rows and then remove the zero rows we obtain the matrix in Figure 4.18.

The matrix obtained after merging blocks $(1,1)$ and B^* of Figure 4.18 is detailed in Figure 4.20, with the columns and rows rearranged. This matrix has full rank. Hence, the matrix of Figure 4.18 has full rank and we have $K(2|E_{NR}| + |E_R|)$ K–RPP solutions affinely independent satisfying inequality (4.42) as an equality. □

p–connectivity inequalities

Let $\{S_0, \dots, S_p\}$ be a partition of V. Assume that $1 \in S_d$, $d \in \{0, \dots, p\}$ and consider one edge $e_j \in E(S_j)$ for every $j \in \{0, \dots, p\} \setminus \{d\}$. Since the p–connectivity inequality (4.37) is valid for the MBCPP on G, from Theorem 4.2.1, the following inequality

$$x^k(\delta_R(S_0)) + (x^k + y^k)(\delta_{NR}(S_0)) + 2 \sum_{1 \leq r < t \leq p} x^k(S_r : S_t) \geq 2 \sum_{i=0, i \neq d}^{p} x^k_{e_i}, \qquad (4.43)$$

for each drone k, is valid for the K–RPP and will be referred to as *p–connectivity inequality.*

Theorem 4.2.20. *p–connectivity inequalities* (4.43) *are facet–inducing for K–*RPP(G) *if subgraphs $(S_i, E_{NR}(S_i))$, $i = 0, \dots, p$, are 3–edge connected, $|(S_0 : S_i)| \geq 2$, $\forall\, i = 1, \dots, p$, the graph induced by $V \setminus S_0$ is connected, and all the edges $e_j \in E(S_j)$ are required edges.*

Proof: Let $z_1, z_2, \dots, z_{m+1}$ and $t_{(a)}$, $a \in E_R$, the 1–RPP tours from Note 4.2.11.1. Consider the graph G^* obtained by deleting from G the required edges in $\delta_R(S_i)$ for all i (if any). Graphs G and G^* have the same set of non–required edges and, given that the hypothesis of Theorem 4.2.9 are fulfilled, there are m affinely independent 1–RPP tours $w^*_1, w^*_2, \dots, w^*_m$ on G^*, all of them traversing the required edges in G^* and satisfying

$$(x + y)(\delta(S_0)) + 2 \sum_{1 \leq r < t \leq p} x(S_r : S_t) = 2p.$$

Given that all the w^*_i traverse each edge $e_j \in E(S_j)$ because is a required edge, they also satisfy

$$(x + y)(\delta(S_0)) + 2 \sum_{1 \leq r < t \leq p} x(S_r : S_t) = 2 \sum_{i=0, i \neq d}^{p} x_{e_i}$$

Each w_j^* is transformed into a tour on G, w_j, by adding to it an extra entry x_e for each removed edge in $\delta_R(S_i)$ for all i, with value $x_e = 0$. The resulting tours $w_1, w_2, \dots, w_m$ on G are also affinely independent, traverse all the required edges except those in sets $\delta_R(S_i)$, and satisfy

$$x(\delta_R(S_0)) + (x+y)(\delta_{NR}(S_0)) + 2 \sum_{1 \le r < t \le p} x(S_r : S_t) = 2 \sum_{i=0, i \ne d}^{p} x_{e_i} \quad \big(= 2p\big).$$

Furthermore, we can assume that one of the w_i above, say $w_1 = (x_1, y_1)$, traverses twice a given non–required edge in each set $(S_0 : S_i)$.

We build the K–RPP solutions depicted in the first three and last two block rows in the matrix shown in Figure 4.17, where vectors z_i, $t_{(a)}$ are the same but vectors w_i are different (although denoted equal).

For each required edge a we define a vector $t^*_{(a)}$ obtained from w_1 that also satisfies the inequality as an equality as follows:

- For each $a \in E_R(S_i)$, $a \ne e_i$, the vector $t^*_{(a)}$ is obtained from w_1 by replacing the traversal of a with the traversal of a path joining its two endpoints formed with edges in $E_{NR}(S_i)$.
- For each edge $e_i \in E_R(S_i)$, the vector $t^*_{(e_i)}$ is obtained from w_1 by removing all the traversals in $(S_0 : S_i)$ and in $E(S_i)$. Note that this vector satisfies

$$x(\delta_R(S_0)) + (x+y)(\delta_{NR}(S_0)) + 2 \sum_{1 \le r < t \le p} x(S_r : S_t) = 2 \sum_{i=0, i \ne d}^{p} x_{e_i} \quad \big(= 2(p-1)\big).$$

- For each $a \in (S_0 : S_i)_R$, the vector $t^*_{(a)}$ is obtained from w_1 by replacing the second traversal of the non–required edge in $(S_0 : S_i)$ traversed twice by w_1 with the traversal of a and a path joining its two endpoints formed with edges in $E_{NR}(S_0) \cup E_{NR}(S_i)$.
- For each $a \in (S_i : S_j)_R$, the vector $t^*_{(a)}$ is obtained from w_1 by adding one copy of a and removing a copy of each non–required edge in $(S_0 : S_i)$ and $(S_0 : S_j)$ traversed twice by w_1 and the traversal of the non–required edges in the paths joining their endpoints.

If drone k performs $t^*_{(a)}$ and the remaining drones perform z_1, we obtain the K–RPP solutions depicted as the rows of the fourth block rows in the matrix shown in Figure 4.17.

If $K \ge 3$ (for the case $K = 2$, the argument is similar to that in the proof of Theorem 4.2.18), all the rows in the matrix depicted in Figure 4.17 represent K–RPP solutions satisfying the p–connectivity inequality (4.43) as an equality. If we subtract the first row from all the other rows and then remove the zero rows we obtain a matrix similar to that in Figure 4.18. The matrix obtained after merging blocks $(1,1)$ and B^* of Figure 4.18 is detailed in Figure 4.21, with the columns and rows rearranged. This matrix has full rank and, therefore, also the matrix of Figure 4.18 has full rank. Hence, we have $K(2|E_{NR}| + |E_R|)$ K–RPP solutions affinely independent satisfying inequality (4.43) as an equality and it defines a facet of K–RPP(G). $\square$

$$
\begin{array}{c|c|c|c|c|c}
E_{NR} & (S_i:S_j)_R & E_R(S_0) & E_R(S_1) & \dots & E_R(S_p) \\
\begin{matrix}(w_2-w_1)_{NR}\\ \cdots \\ (w_m-w_1)_{NR}\end{matrix} & 0 & 0 & 0 & \cdots & 0 \\
\hline
* & \begin{matrix}1 & & \\ & \ddots & \\ & & 1\end{matrix} & 0 & 0 & \cdots & 0 \\
\hline
* & 0 & \begin{matrix}-1 & & \\ & \ddots & \\ & & -1\end{matrix} & 0 & \cdots & 0 \\
\hline
* & 0 & 0 & \begin{matrix}-1 & & & \\ & \ddots & & \\ & & -1 & \\ -1 & \dots & -1 & -1\end{matrix} & \cdots & 0 \\
\hline
* & 0 & 0 & 0 & \cdots & 0 \\
\hline
* & 0 & 0 & 0 & \cdots & \begin{matrix}-1 & & & \\ & \ddots & & \\ & & -1 & \\ -1 & \dots & -1 & -1\end{matrix}
\end{array}
$$

Figure 4.21: A submatrix of the matrix in Figure 4.18 for p–connectivity inequalities

K–C inequalities

The name of K–C inequalities is motivated by the number of subsets into which V is partitioned, which is usually denoted by K. In order to avoid confusion here with the number of vehicles, we will use the letter Q to denote the number of sets in a K–C partition. Let $\{S_0, \dots, S_Q\}$, with $Q \geq 3$, be a partition of V. We assume that the depot $1 \in (S_0 \cup S_Q)$ and $E(S_i) \neq \emptyset$, for all $i = 1, 2, \dots, Q-1$. We select one edge $e_i \in E(S_i)$ for each i. Let $F \subseteq (S_0 : S_Q)$ be a set of edges, with $|F| \geq 2$ and even. For each drone k, the following inequalities

$$
\begin{aligned}
&(Q-2)\, x^k\Big((S_0:S_Q)_R \backslash F\Big) - (Q-2)\, x^k(F_R) + \\
&+ (Q-2)\,(x^k - y^k)\Big((S_0:S_Q)_{NR} \backslash F\Big) - (Q-2)\,(x^k - y^k)(F_{NR}) + \\
&+ \sum_{\substack{0 \leq i<j \leq Q \\ (i,j)\neq(0,Q)}} (j-i)\, x^k(S_i:S_j)_R + \sum_{\substack{0 \leq i<j \leq Q \\ (i,j)\neq(0,Q)}} \Big((j-i)\, x^k(S_i:S_j)_{NR} + (2-j+i)\, y^k(S_i:S_j)_{NR}\Big) \geq \\
&\geq 2 \sum_{i=1}^{Q-1} x^k_{e_i} - (Q-2)|F|. \qquad (4.44)
\end{aligned}
$$

will be referred to as *K–C inequalities*. If the depot $1 \in S_d$, $d \in \{1, \dots, Q-1\}$, the RHS of the K–C inequalities is:

$$\geq 2 \sum_{\substack{i=1 \\ i \neq d}}^{Q-1} x^k_{e_i} + 2 - (Q-2)|F|. \tag{4.45}$$

From Theorem 4.2.1, K–C inequalities (4.44) and (4.45) are valid for the K–RPP because they are obtained from the corresponding K–C inequalities for the MBCPP (see Corberán et al., 2013) after removing the y_e variables for all the required edges e.

Theorem 4.2.21. *K–C inequalities* (4.44) *and* (4.45) *are facet–inducing for K–*RPP(G) *if subgraphs $(S_i, E_{NR}(S_i))$, $i = 0, \dots, Q$, are 3–edge connected, $|(S_i : S_{i+1})| \geq 2$ for $i = 0, \dots, Q-1$, and $|F_R| \geq 2$ and $e_i \in E_R(S_i)$ for all i.*

Proof: We make the proof only for inequalities (4.44) since the proof for inequalities (4.45) is analogous. Let $z_1, z_2, \dots, z_{m+1}$ and $t_{(a)}$, $a \in E_R$, be the 1–RPP tours from Note 4.2.11.1. Consider the graph G^* obtained by deleting from G the required edges in $\delta_R(S_i)$, for all $i = 1, \dots, Q-1$, and the required edges in $(S_0 : S_Q) \setminus F$. Graphs G and G^* have the same set of non–required edges and, given that the hypothesis of Theorem 4.2.10 are fulfilled, there are m affinely independent 1–RPP tours $w^*_1, w^*_2, \dots, w^*_m$ on G^*, all of them traversing the required edges in G^*, and satisfying inequality (4.40) with $|\mathcal{R}| = Q-1$ as an equality:

$$(Q-2)(x-y)\Big((S_0 : S_Q) \setminus F\Big) - (Q-2)(x-y)(F_{NR}) + \\ + \sum_{\substack{0 \leq i < j \leq Q \\ (i,j) \neq (0,Q)}} \Big((j-i)\, x(S_i : S_j) + (2-j+i)\, y(S_i : S_j)\Big) = 2(Q-1) - (Q-2)|F_{NR}|.$$

Given that all the w^*_j traverse each edge $e_i \in E(S_i)$, they also satisfy

$$(Q-2)(x-y)\Big((S_0 : S_Q) \setminus F\Big) - (Q-2)(x-y)(F_{NR}) + \\ + \sum_{\substack{0 \leq i < j \leq Q \\ (i,j) \neq (0,Q)}} \Big((j-i)x(S_i : S_j) + (2-j+i)y(S_i : S_j)\Big) = 2\sum_{i=1}^{Q-1} x_{e_i} - (Q-2)|F_{NR}|.$$

Each w^*_j is transformed into a tour on G, w_j, by adding to it an extra entry x_e for each removed edge in $\delta_R(S_i)$, for all $i = 1, \dots, Q-1$, and in $(S_0 : S_Q)_R \setminus F$, with value $x_e = 0$. The resulting tours $w_1, w_2, \dots, w_m$ on G are also affinely independent, traverse all the required edges except those in sets $\delta_R(S_i)$ and in $(S_0 : S_Q)_R \setminus F$, and satisfy

$$(Q-2)\, x\Big((S_0 : S_Q)_R \backslash F\Big) - (Q-2)\, x(F_R) + \\ + (Q-2)\, (x-y)\Big((S_0 : S_Q)_{NR} \backslash F\Big) - (Q-2)\, (x-y)(F_{NR}) + \\ + \sum_{\substack{0 \leq i < j \leq Q \\ (i,j) \neq (0,Q)}} (j-i)\, x(S_i : S_j)_R + \sum_{\substack{0 \leq i < j \leq Q \\ (i,j) \neq (0,Q)}} \Big((j-i)\, x(S_i : S_j)_{NR} + (2-j+i)\, y(S_i : S_j)_{NR}\Big) = \\ = 2\sum_{i=1}^{Q-1} x_{e_i} - (Q-2)|F|, \tag{4.46}$$

because each w_j traverses each edge in F_R and, hence, $x(F_R) = |F_R|$. Furthermore, we can assume that one of the w_j above, say $w_1 = (x_1, y_1)$, traverses once each edge in F, and traverses twice a given non–required edge in each set $(S_j : S_{j+1})$ for $j = 0, \dots, Q-2$.

We build the K–RPP solutions depicted in the first three and last two block rows in the matrix shown in Figure 4.17, where vectors z_i, $t_{(a)}$ are the same but vectors w_i are different (although denoted equal).

For each edge $a \in E_R$ we define a vector $t^*_{(a)}$ obtained from w_1 and also satisfying equation (4.46) as follows:

- For each $a \in E_R(S_i)$, $a \neq e_i$, $i = 0, 1, \dots, Q$, the vector $t^*_{(a)}$ is obtained from w_1 by replacing the traversal of a with the traversal of a path joining its two endpoints formed with edges in $E_{NR}(S_i)$.
- For each $a = e_i \in E_R(S_i)$, $i \neq Q-1$, the vector $t^*_{(a)}$ is obtained from w_1 by removing all the traversals in $E(S_i)$, $(S_{i-1} : S_i)$, and $(S_i : S_{i+1})$, and adding two copies of a non–required edge in $(S_{Q-1} : S_Q)$. For the edge $a = e_{Q-1}$, the vector $t^*_{(a)}$ is obtained from w_1 by removing all the traversals in $E(S_{Q-1})$ and in $(S_{Q-2} : S_{Q-1})$.
- For each $a \in (S_j : S_{j+1})_R$, $j = 0, \dots, Q-1$, the vector $t^*_{(a)}$ is obtained from w_1 by replacing the second traversal of the non–required edge traversed twice in $(S_j : S_{j+1})$ with the traversal of a and a path joining its two endpoints formed with edges in $E_{NR}(S_j) \cup E_{NR}(S_{j+1})$.
- For each $a \in (S_i : S_j)_R$, $|i-j| > 1$ the vector $t^*_{(a)}$ is obtained from w_1 by adding one copy of a and removing the second traversal of the non–required edge traversed twice in $(S_i : S_{i+1}), \dots, (S_{j-1} : S_j)$ and adding the traversal of the non–required edges in some paths in $E(S_i), \dots, E(S_j)$.
- For each $a \in F_R$, the vector $t^*_{(a)}$ is obtained from w_1 by removing a and all the second traversals of the non–required edges traversed twice in sets $(S_j : S_{j+1})$, $j = 0, \dots, Q-2$, and adding a non–required edge in $(S_{Q-1} : S_Q)$ and the non–required edges in some paths in $E(S_0), E(S_1) \dots, E(S_Q)$.
- For each $a \in (S_0 : S_Q)_R \setminus F$, the vector $t^*_{(a)}$ is obtained from w_1 by adding a, removing all the second traversals of the non–required edges traversed twice in sets $(S_j : S_{j+1})$, $j = 0, \dots, Q-2$, and adding a non–required edge in $(S_{Q-1} : S_Q)$ and the non–required edges in some paths in $E(S_0), E(S_1) \dots, E(S_Q)$.

If drone k performs $t^*_{(a)}$ and the remaining drones perform z_1 we obtain the K–RPP solutions depicted as the rows of the fourth block rows in the matrix shown in Figure 4.17.

If $K \geq 3$ (for the case $K = 2$, the argument is similar to that in the proof of Theorem 4.2.18), all the rows in the matrix depicted in Figure 4.17 represent K–RPP solutions satisfying the K–C inequality as an equality. If we subtract the first row from all the other rows and then remove the zero rows we obtain a matrix similar to that in Figure 4.18. The matrix obtained after merging blocks $(1, 1)$ and B^* of Figure 4.18 is detailed in Figure 4.22, with the columns and rows rearranged. For the sake of simplicity, the elements in the block corresponding to edges in $E_R(S_0 \cup S_Q)$ are ± 1, representing 1 for the edges in F_R, and -1 for the edges in $E_R(S_0)$, $E_R(S_Q)$, and $(S_0 : S_Q)_R \setminus F$. This matrix has full rank. Hence, we have $K(2|E_{NR}| + |E_R|)$ K–RPP solutions affinely independent satisfying the K–C inequality (4.44) as an equality. $\square$

$$
\begin{array}{c|c|c|c|c|c}
E_{NR} & (S_i:S_j)_R & E_R(S_0 \cup S_Q) & E_R(S_1) & \dots & E_R(S_{Q-1}) \\
\begin{matrix} (w_2-w_1)_{NR} \\ \cdots \\ (w_m-w_1)_{NR} \end{matrix} & 0 & 0 & 0 & \cdots & 0 \\
\hline
* & \begin{matrix} 1 & & \\ & \ddots & \\ & & 1 \end{matrix} & 0 & 0 & \cdots & 0 \\
\hline
* & 0 & \begin{matrix} \pm 1 & & \\ & \ddots & \\ & & \pm 1 \end{matrix} & 0 & \cdots & 0 \\
\hline
* & 0 & 0 & \begin{matrix} -1 & & & \\ & \ddots & & \\ & & -1 & \\ -1 & \dots & -1 & -1 \end{matrix} & \cdots & 0 \\
\hline
* & 0 & 0 & 0 & \cdots & 0 \\
\hline
* & 0 & 0 & 0 & \cdots & \begin{matrix} -1 & & & \\ & \ddots & & \\ & & -1 & \\ -1 & \dots & -1 & -1 \end{matrix}
\end{array}
$$

Figure 4.22: A submatrix of the matrix in Figure 4.18 for K–C inequalities

4.2.3 Other valid inequalities for the LC K–RPP

Although we have removed the constraints (4.3), which limit the length of each route k, to carry out the polyhedral study of the $K-$RPP, they have to be taken into account when solving the LC $K-$RPP. Based on these constraints, we present here some sets of valid inequalities for the LC K–RPP, called *max–length inequalities.*

Let $F \subseteq E_R$ be a subset of required edges. Consider the general routing problem defined on graph G, with required edges set F and required vertex 1 (the depot), if it is not incident with an edge in F. Recall that the GRP consists of finding a minimum cost tour traversing the edges of F at least once and visiting the depot. Let $\text{GRP}(F)$ be its optimal value (or a lower bound of it). If $\text{GRP}(F) > L$, then the following inequalities are valid for the LC K–RPP:

$$x^k(F) \leq |F| - 1, \quad \forall k = 1, \dots, K, \tag{4.47}$$

which indicate that a single vehicle cannot service all the edges in F.

Let S be the set of vertices incident with the edges in F, and suppose that $1 \notin S$. If $\text{GRP}(F) > L$, then at least two vehicles must enter in S, and we have

$$\sum_{k=1}^{K} \left(x^k(\delta_R(S)) + (x^k + y^k)(\delta_{NR}(S)) \right) \geq 4. \tag{4.48}$$

Moreover, to force two *different* vehicles to enter S, instead of a single vehicle entering S twice, we have the following inequalities:

$$\sum_{k \neq k'} \big(x^k(\delta_R(S)) + (x^k + y^k)(\delta_{NR}(S))\big) \geq 2, \qquad \forall k' = 1, \ldots, K. \tag{4.49}$$

The above inequalities can be easily generalized to any value $p = \lceil \frac{\text{GRP}(F)}{L} \rceil > 1$ if $\text{GRP}(F) > (p-1)L$.

Chapter 5

An iterative algorithm for the length constrained K–DRPP

The length constrained K–drones rural postman problem (LC K–DRPP) was defined in Chapter 3, and a matheuristic algorithm was proposed for its solution. This chapter presents a new mathematical approach to deal with the problem.

Given an LC K–DRPP instance, if the number of intermediate points into which each line is approximated is large, the size of the resulting LC K–RPP instance is huge and, therefore, very hard to solve. However, if we only incorporate a few of these points, we will get an smaller (and easier to address) K–RPP instance. Hence, it is necessary to implement a procedure that iteratively generates LC K–RPP instances by approximating each line by a polygonal chain with few but significant points and segments, and then solving such instances with a competitive exact procedure. This is the idea behind the algorithm we present here.

Based on the formulation proposed in the previous chapter for the LC K–RPP and the polyhedral study developed, we have designed a new branch–and–cut algorithm for the LC K–RPP that incorporates the separation of the valid inequalities proposed there. We describe this algorithm in Section 5.1. This branch–and–cut algorithm is integrated into a solution procedure for the LC K–DRPP with a sequential scheme that iteratively solves instances of the LC K–RPP which are generated by adding and removing some intermediate points (adding those considered more 'promising' and removing the unused ones) in a way that is a refinement of the algorithm presented in Chapter 3. This iterative algorithm is described in Section 5.2. We provide in Section 5.3 some computational results showing the effectiveness of the new algorithm, comparing its performance with that of the matheuristic proposed in Chapter 3.

5.1 A branch–and–cut algorithm for the LC K–RPP

In this section, we describe the branch–and–cut algorithm we have designed and implemented for the LC K–RPP. This method is based on a cutting–plane algorithm that incorporates separation procedures for the inequalities presented in Chapter 4.

The initial LP relaxation contains the inequalities

$$\sum_{e\in E_R} c_e^s x_e^k + \sum_{e\in E_{NR}} c_e\big(x_e^k + y_e^k\big) \le L, \qquad \forall k = 1,\dots,K, \tag{4.3}$$

$$\sum_{k=1}^{K} x_e^k \ge 1, \qquad \forall e \in E_R, \tag{4.4}$$

$$x_e^k \ge y_e^k, \qquad \forall e\in E_{NR}, \;\; \forall k = 1,\dots,K, \tag{4.5}$$

from the formulation (F2), the bounds on the variables, and a parity inequality (4.42) with $F = \delta(v)$ for each odd–degree vertex v and for each drone k.

Moreover, in order to avoid equivalent solutions produced by the interchange of routes between two (or more) vehicles, the following *symmetry breaking* inequalities are also added to the LP. Let us assume the required edges are ordered as $e_1, e_2, \dots, e_{|E_R|}$ (for example, in descending order according to the distances between them and the depot). Then,

$$x_{e_1}^1 = 1, \tag{5.1}$$

$$x_{e_i}^k \le \sum_{j=1}^{i-1} x_{e_j}^{k-1}, \qquad \forall k = 2,\dots,K, \quad \forall i \ge 2, \text{ and} \tag{5.2}$$

$$x_{e_i}^k = 0, \qquad \forall k = i+1,\dots,K, \quad \forall i = 1,\dots,|E_R|-1. \tag{5.3}$$

Equality (5.1) forces vehicle 1 to traverse (and service) required edge e_1. Inequalities (5.2) state that if vehicle k services the required edge e_i, then at least one required edge in $\{e_1,\dots,e_{i-1}\}$ (i.e., a previous edge in the given ordering) has to be serviced by vehicle $k-1$. Equations (5.3) prevent a required edge e_i, for $i = 1,\dots,|E_R|-1$, from being serviced by a vehicle whose index is larger than i.

In addition, to prune the search tree, we use a bound obtained by the first phase of the matheuristic algorithm proposed in Section 3.3.

5.1.1 Separation algorithms

We describe here the separation algorithms used to identify violated inequalities of the following types: connectivity inequalities (4.2), parity inequalities (4.42), p–connectivity inequalities (4.43), K–C inequalities (4.44) and (4.45), and max–length inequalities (4.47), (4.48) and (4.49).

Given a fractional solution $(\bar{x}^k, \bar{y}^k)$, for a vehicle k, of the current LP and a parameter $\varepsilon \ge 0$, we will use two support graphs, denoted by $G_+^k(\varepsilon)$ and $G_-^k(\varepsilon)$, which are the graphs induced by the edges $e \in E$ such that $\bar{x}^k(e) + \bar{y}^k(e) > \varepsilon$ and $\bar{x}^k(e) - \bar{y}^k(e) > \varepsilon$, respectively, plus the depot, if necessary.

Connectivity inequalities

For each vehicle k, connectivity inequalities (4.2) can be exactly separated in polynomial time with the following well–known algorithm. For each edge f such that $\bar{x}^k(f) > 0$, compute the minimum–weight cut in graph $G_+^k(0)$ separating edge f from the depot. If the weight of this cut is less than $2\bar{x}^k(f)$, then the corresponding inequality (4.2) is violated.

Although polynomial, this exact algorithm is very time consuming and often produces many violated inequalities that are very similar to each other. Therefore, we avoid studying the edges close to an edge f for which a violated connectivity inequality has already been found. Thus, the number of calculated minimum cuts and inequalities added to the LP are reduced.

Two heuristic algorithms have also been implemented in order to find violated connectivity inequalities. The first one computes the connected components of the graph $G_+^k(\varepsilon)$. The inequality (4.2) associated with each component not containing the depot and the edge f in it with maximum $\bar{x}^k(f)$ is checked for violation. The second algorithm is based on reducing the size of the graph on which the minimum cuts are computed, by shrinking the connected components of the graph induced by edges with $\bar{x}^k(e) \geq 1 - \varepsilon$ into a single vertex each. Then, we compute all the minimum cuts between the vertex corresponding to the component containing the depot and the remaining ones. The corresponding connectivity inequalities are checked for violation.

Parity inequalities

If we replace $x - y$ by x in the parity inequalities (4.42) for a vehicle k, we obtain

$$x^k(\delta(S)\backslash F) \geq x^k(F) - |F| + 1, \quad \forall S \subset V, \quad \forall F \subset \delta(S) \text{ with } |F| \text{ odd},$$

which are the well–known cocircuit inequalities presented in Ghiani and Laporte (2000). They can be exactly separated in polynomial time with an algorithm based on the computation of odd minimum cuts, which can be done with the classical Padberg–Rao procedure (Padberg and Rao, 1982) or with the improved one by Letchford et al. (2008).

For the special case where $S = \{v\}$, there is an exact and simple procedure to define the set $F \subseteq \delta(v)$ that corresponds to the most violated parity inequality (see Ghiani and Laporte, 2000).

We also use a heuristic algorithm based on the computation of the connected components of the support graph $G_-^k(\varepsilon)$ for each vehicle k. For each cut–set obtained from a connected component, the set F is found by applying the same procedure as for $S = \{v\}$.

p–connectivity inequalities

For each vehicle k, we use a heuristic algorithm similar to the one proposed in Corberán et al. (2013) for the MBCPP. The algorithm starts by searching for a cut–set $(S, V\backslash S)$ corresponding to a tight connectivity inequality (4.2) among those obtained with the connectivity separation procedures. Let us suppose that $1 \in V\backslash S$ and let $S_0 = V\backslash S$. Then we compute the connected components in the subgraph induced in $G(S)$ by the edges $e \in E(S)$ with $\bar{x}^k(e) \geq 1 - \varepsilon$, where ε is a given parameter. For each pair C_i, C_j of such components, we compute

$$s_{ij} = 2\bar{x}^k(V_i, V_j) - 2\min\{\bar{x}^k(e_i), \bar{x}^k(e_j)\},$$

where V_r is the set of vertices in the component C_r and e_r is the edge in C_r with the highest value of $\bar{x}^k(e)$. The value s_{ij} represents the savings in the left-hand side of the inequality obtained after shrinking components C_i and C_j. We iteratively shrink the components that maximize s_{ij} while $s_{ij} > 0$. This procedure defines sets $S_1, \dots, S_p$. The p–connectivity inequality associated with this partition is checked for violation.

K–C inequalities

For each vehicle k, we have implemented a heuristic algorithm that is an adaptation of the one proposed in Corberán et al. (2001) for the general routing problem. In that problem, the required edges determine the way in which set V is partitioned in $S_0, \dots, S_K$. Here, we partition V in the same way but considering as required edges those with $\bar{x}^k(e) \geq 1 - \varepsilon$, which will define the set F.

Max–length inequalities

In order to separate the max–length inequalities proposed in Section 4.2.3, we use two heuristic algorithms. The first heuristic looks for violated inequalities (4.47). It tries to cut fractional solutions in which, for a vehicle k, several x_e^k variables, for $e \in E_R$, take values close to 1 and another one takes a value close to 0.5. Let $\{e_1, e_2, \dots, e_m\}$ be a set of required edges such that $\bar{x}_{e_1}^k \geq \bar{x}_{e_2}^k \geq \dots \geq \bar{x}_{e_m}^k \geq 0.5$. We define $F = \{e_1, e_2, \dots, e_f\}$, where f is the maximal number such that $\bar{x}^k(F) > |F| - 1 + \varepsilon$ (initially we set $\varepsilon = 0.5$), and we call "potential edges" the remaining $\{e_{f+1}, \dots, e_m\}$. We check if $\text{GRP}(F)$ is greater than L and, therefore, the corresponding inequality (4.47) is violated. Otherwise, for each potential edge e^*, we iteratively consider the set $F^* = F \cup \{e^*\}$ and check if $\text{GRP}(F^*)$ is greater than L. Finally, if no violated inequality has been found for any set F^*, we set $\varepsilon = 0$ and repeat the process. For each subset F (or F^*) whose corresponding inequality (4.47) is violated, we look for the cutset of minimum weight between the depot and the edges in F and the corresponding max–length inequalities (4.48) and (4.49) are checked for violation.

The second heuristic looks for inequalities (4.48). We first consider the "aggregate" solution $\bar{x}_e = \sum_{k=1}^K \bar{x}_e^k$ and $\bar{y}_e = \sum_{k=1}^K \bar{y}_e^k$. The procedure starts by selecting the vertex $i \neq 1$ farthest from the depot such that the maximum flow from 1 to i is less than $2K$. Let $\delta(S)$ be the corresponding minimum weight cut–set. Then, a sequence of vertices is iteratively added to S in such a way that $\sum_{k=1}^K \bar{x}^k(\delta_R(S)) + \sum_{k=1}^K (\bar{x}^k + \bar{y}^k)(\delta_{NR}(S))$ is minimum for the resulting set S. For each subset S generated, we compute the minimum number of vehicles needed to service all the edges in $E_R(S)$ by solving the associated GRP, and the corresponding inequality (4.48) is checked for violation. If a violated constraint (4.48) is found, at least one of the inequalities (4.49) is also violated and it is added. Furthermore, the corresponding inequality (4.47) is also checked for violation.

Given a set of edges F, the value of $\text{GRP}(F)$ is computed by solving the corresponding GRP with the branch–and–cut algorithm described in Corberán et al. (2007). To minimize the number of GRPs solved, two lists containing the already studied sets F are managed.

5.1.2 The cutting–plane algorithm

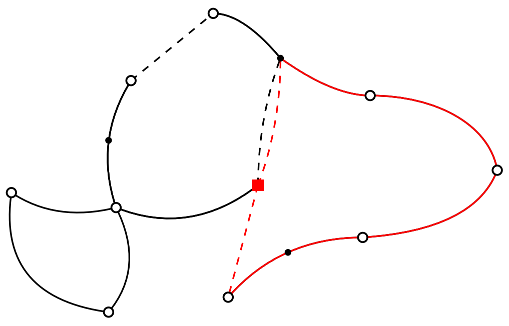
The cutting–plane algorithm applies, at each iteration, the separation procedures described above in this specific order:

1. The first heuristic algorithm for connectivity inequalities is applied with $\varepsilon = 0, 0.25, 0.5$. The value of ε is increased only if the previous one results in no violated inequalities found.
2. If no violated inequalities are found so far, the second connectivity heuristic is executed for all the values $\varepsilon = 0, 0.1, 0.2, 0.3, 0.4$. For each tight cut–set obtained, the p–connectivity heuristic separation algorithm is invoked with $\varepsilon \in \{0, 0.15, 0.3\}$.
3. The exact parity separation procedure for single vertices is applied.
4. The heuristic procedure for parity inequalities with $\varepsilon = 0, 0.25, 0.5$. The different values of ε are used only if no violated inequalities are obtained with the previous ones.
5. If no violated connectivity nor parity inequalities have been found for a given vehicle k, the separation algorithm for K–C inequalities for this vehicle is applied with parameter $\varepsilon = 0, 0.2$.
6. Only at the root node and nodes whose depth in the search tree is a multiple of 3, we apply the max–length separation algorithms.
7. Only at the root node and if no violated connectivity, parity nor K–C inequalities have been found, we apply the exact procedure for parity inequalities.
8. Finally, only at the root node and if no violated connectivity, parity nor K–C inequalities have been found, we apply the exact procedure for connectivity inequalities. Again, for each tight cut–set obtained, the p–connectivity heuristic separation algorithm is invoked with $\varepsilon \in \{0, 0.15, 0.3\}$.

5.2 Iterative algorithm for the LC K–DRPP

We describe in this section the algorithm we have developed for solving the LC K–DRPP. At each iteration, the procedure first selects some intermediate points from the original lines and generates the corresponding LC K–RPP instance that includes these points as vertices. Then, this instance is solved with the branch–and–cut algorithm described in Section 5.1, and its solution is used to define another LC K–RPP instance by adding and removing some intermediate points. It is important to point out that although the proposed method embebes an exact algorithm at each iteration, the global algorithm is a heuristic algorithm for the LC K–DRPP.

Let V_o denotes the set of endpoints of the original lines in the LC K–DRPP instance. The procedure begins by applying the matheuristic algorithm described in Section 3.3 for the LC K–DRPP, and also solving with the branch–and–cut algorithm presented in Section 5.1 the LC K–RPP(0) instance, that is, the LC K–RPP instance in which each original line is approximated by only one (required) edge, without intermediate points.


(a) LC K–RPP[i] solution with $K = 2$ drones

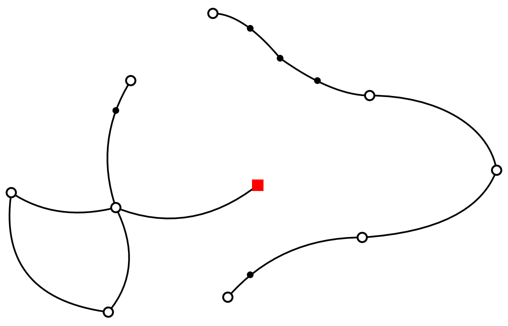

(b) LC K–RPP[$i + 1$] instance with vertex set $V_0 \cup W_{i+1}$

Figure 5.1: Illustration of the procedure that adds intermediate vertices

From these two solutions obtained with the matheuristic and the branch and cut, we generate a new LC K–RPP instance in the following way. Let W_0 be the set of vertices incident with non–required edges used by any of these two solutions. Note that all the vertices in W_0 obtained from the solution of the LC K–RPP(0) instance are vertices of V_o, while some vertices in W_0 obtained from the solution provided by the matheuristic can be intermediate points.

From W_0, we define another set W_1 of intermediate points to generate the next LC K–RPP instance as follows. For each line incident with an endpoint in W_0, we add to W_1 the intermediate point that splits the line in two segments of the same length. Furthermore, all the vertices in $W_0 \setminus V_o$ are added to W_1 to guarantee that the solution from the heuristic is a feasible solution of the new instance.

The instance with vertex set $V_o \cup W_1$, called LC K–RPP[1], is generated with the segments joining vertices in $V_o \cup W_1$ as required edges and it is solved with the branch–and–cut algorithm.

Given the solution of instance LC K–RPP[i], $i \geq 1$, with vertex set $V_o \cup W_i$, we generate the instance LC K–RPP[i+1] with vertex set $V_o \cup W_{i+1}$, where W_{i+1} is defined as follows. We initialize W_{i+1} with the vertices of W_i incident with a non–required edge used by the solution. Then, for each required edge of LC K–RPP[i] having at least one endpoint incident with a non–required edge in the solution, we add to W_{i+1} the intermediate point that splits this edge into two parts of equal length. Note that the vertices of W_i that are not used in the solution are not included in the new instance.

Figure 5.1b illustrates the LC K–RPP[$i + 1$] instance with vertex set $V_0 \cup W_{i+1}$ obtained from the LC K–RPP[i] optimal solution depicted in Figure 5.1a with 2 drones and vertex set $V_0 \cup W_i$. In both figures, the square node represents the depot, white nodes represent set V_0, and black nodes are the vertices in W_i and W_{i+1}. Solid lines correspond to required edges, while dashed lines represent non–required ones.

This procedure is iteratively applied up to $i = 4$ and while the computing time does not exceed two hours.

Although with this iterative procedure the size of the vertex set of each LC K–RPP instance does not increase much, the number of non–required edges is still huge, since they induce a complete graph. In order to reduce the number of non–required edges, we apply the following *preprocessing* procedure to each instance LC K–RPP[i], $i \geq 1$:

First, all the non–required edges (i, j) such that there is a vertex k with $c_{ij} \geq 0.99(c_{ik} + c_{kj})$ are removed. Moreover, if c_{max} is the length of the longest non–required edge, we remove those edges (i, j) such that $c_{ij} > c_{max}/3$ and there is a vertex k with $c_{ij} \geq 0.95(c_{ik} + c_{kj})$. However, for $i \geq 1$, if a non–required edge is used in the solution of LC K–RPP[$i - 1$], it is not removed in order to guarantee that the solution of the new instance is not worse than the previous one.

Since we have removed some non–required edges in the previous preprocessing, it is possible that the solution of an instance contains two adjacent non–required edges (i, k), (k, j). If this happens, we replace them with the non–required edge (i, j) if the resulting solution is not disconnected.

5.3 Computational experiments

We present here the computational experiments carried with the iterative algorithm. The procedure has been tested on the two sets of LC K–DRPP instances presented in Chapter 3, which were based on the ones from Campbell et al. (2018) for the Drone RPP. The first set consists of 30 *random* instances, while the second one, called *even*, contains 15 instances obtained by modifying the set of required edges of some of the random instances to reduce the number of odd–degree vertices. These Drone RPP instances have between 18 and 92 lines and between 22 and 83 nodes (see Table 3.1). Five different values for the length limit L were generated in Chapter 3 for each instance in such a way that the number of drones for each one ranges from 2 to 6, resulting a total of 225 instances. Recall that all these instances are available at http://www.uv.es/plani/instancias.htm.

The algorithms have been implemented in C++ and all the tests have been run on an Intel Core i7 at 3.4 GHz with 32 GB 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.

5.3.1 Computational results of the iterative algorithm

In this section we present the results obtained with the iterative algorithm on the 225 LC K–DRPP instances with a time limit of two hours. Table 5.1 summarizes these results organized by set of instances and number of vehicles. Columns 4, 5, 6, and 7 report the average values obtained for all the instances of each group. The "Imp(%)" column shows the percentage improvement of the deadheading cost of the best solution found by the algorithm with respect to the solution provided by the matheuristic proposed in Chapter 3, and column "TimeH" reports the average computing time in seconds used by the heuristic. Columns "Time" and "♯iter" present the average computing time in seconds and the average number of iterations done by the iterative algorithm, respectively. Column 8 gives the number of instances in which the new algorithm was able to find a better solution than the one given by the matheuristic procedure. The

			All instances					**Improved instances**				
Type	K	♯ inst	Imp(%)	TimeH	Time	♯ iter	♯ imp	Imp(%)	TimeH	Time	♯ iter	♯ itb
R	2	30	0.41	46.2	3815.2	4.2	13	0.87	40.4	2842.1	4.4	3.6
	3	30	2.12	37.4	5253.2	3.2	22	2.65	37.2	5182.3	3.4	2.3
	4	30	3.38	36.5	6310.1	2.5	19	5.33	32.2	5773.5	3.2	2.5
	5	30	1.89	35.6	6959.2	1.9	14	3.77	27.0	6673.4	2.3	1.5
	6	30	0.66	36.9	7130.6	1.8	8	2.47	30.2	6819.1	3.1	2.0
E	2	15	1.05	75.5	2583.2	4.3	13	1.21	42.5	1872.9	4.8	4.1
	3	15	1.44	51.6	5922.3	3.1	10	2.16	26.5	4003.3	4.1	3.4
	4	15	2.07	58.3	6587.9	2.5	9	3.46	22.7	5070.5	3.6	2.8
	5	15	0.62	58.0	5423.2	2.0	5	1.87	19.1	3990.1	3.6	3.4
	6	15	0.38	58.5	6364.0	1.9	5	1.13	12.6	5363.6	3.6	3.4
Total		**225**	1.50				**118**	2.77				

Table 5.1: Improvement of the global algorithm over the matheuristic

next four columns report the same figures as columns 4 to 7, but only for those instances for which a better solution was found. Last column "♯itb" gives the average number of iterations of the global algorithm needed to reach the best solution.

As we can see in Table 5.1, the new algorithm has improved the results provided by the matheuristic in 118 out 225 instances and the average improvement obtained ranges from 0.87% to 5.33%. In the case of the *random* instances, when the number of vehicles is 2, our algorithm improves only 13 out of 30 instances, despite being able to complete the 5 iterations in most of them. This is probably due to the high quality of the solutions provided by the matheuristic algorithm for these instances. Note that with 3, 4, and 5 vehicles, the number of improved solutions increases, as well as the average improvement. The results with 6 vehicles present lower improvements, but in this case this may be due to the iterative algorithm not being able to complete more than 2 iterations on average. Regarding the *even* instances, except those with 2 vehicles, both the number of instances in which the iterative algorithm improves the matheuristic, as well as the percentage of improvement, are smaller. This may be due to the fact that, as pointed out before, *even* instances are more difficult for branch–and–cut algorithms. The improvements obtained by the new algorithm require a considerable running time, but note that the number of iterations needed to reach the best solution is lower than the total number of iterations.

In order to describe with more detail the behavior of the proposed procedure, we present in Table 5.2 the results for a specific instance with 2 to 6 vehicles. Column "Heur" reports the deadheading cost of the solution provided by the matheuristic. Columns labeled 1 to 5 show the deadheading costs of the solutions obtained in the corresponding iteration of the algorithm. All these costs correspond to optimal solutions of the branch–and–cut for the LC K–RPP instances except those marked with an "asterisk", which are associated with the best feasible solution known when the time limit was reached. The total computing time in seconds is reported in the last column. In the instance with 2 vehicles, the algorithm is capable of doing 5 iterations in only 5 minutes, but no improvement is obtained. However, in the instances with 3 and 4 vehicles a better solution is found in each of the 5 iterations. When 5 and 6 vehicles are considered, the two hour time limit is reached without the algorithm being able to complete all 5 iterations, although the last feasible solutions found improve those of the

		iteration number					
K	Heur	**1**	**2**	**3**	**4**	**5**	Time
2	1889.37	1889.37	1889.37	1889.37	1889.37	1889.37	294.0
3	2154.41	2060.28	1989.41	1946.42	1924.66	1913.69	526.2
4	2709.32	2634.49	2459.05	2413.75	2388.51	2377.05	6888.3
5	3369.75	3348.92	3348.91	3348.91*			7200.0
6	4588.93	4626.03	4399.8*				7200.0

Table 5.2: Detailed results on *random* instance DroneRPP77_1

heuristic.

5.3.2 Testing the branch–and–cut algorithm for the LC K–RPP

Although the main goal of this chapter is to find good solutions for the LC K–DRPP, we have studied in depth the LC K–RPP, for which we proposed in Chapter 4 a new formulation and studied its associated polyhedron, and based on this, we have designed and implemented in this chapter a branch–and–cut algorithm. Since the LC K–RPP and its solution have an interest by themselves, we test in this section the performance of the proposed branch and cut (BC2 in what follows). In order to do so, we compare it with that presented in Section 3.2 (BC1 in what follows). Since that branch and cut was run only on the LC K–RPP instances in which each original line is approximated by only one (required) edge without intermediate points, denoted LC K–RPP(0), we compare both branch–and–cut algorithms only on these instances, with a time limit of two hours.

The results are reported in Table 5.3, where columns 4 and 7 show the percentage average gaps of the lower bounds obtained with both procedures with respect to the cost of the best solution known, while columns labeled "♯opt" and "♯ub" report, for both algorithms, the number of instances for which an optimal solution or a feasible solution has been found, respectively.

Overall, the performance of BC2 is very good and clearly superior to that of BC1. In particular, it finds the optimal solution in 137 instances and obtains a feasible solution in 211 out of 225, while the BC1 obtains 74 optimal solutions and 162 feasible ones. Regarding the gaps, in most of the cases the gap obtained by BC2 is about half the gap by BC1. Moreover, in the 53 *random* instances that are optimally solved by both methods, BC2 uses 367.2 seconds on average, while BC1 needs 726.0 seconds. The average times corresponding to the 21 *even* instances solved by both methods are 165.1 and 1391.0 seconds, respectively.

To assess the contribution of the valid inequalities used in the branch and cut proposed in this chapter (BC2), we have compared in Table 5.4 the results obtained by this procedure with a branch and cut using only connectivity inequalities as lazy cuts when the LP solutions are integer and CPLEX default cuts (Basic B&C) on the total 225 instances. Column 1 reports the average percentage gap between the lower bound obtained at the end of the root node and the best solution known, while column 2 reports

			BC1			**BC2**		
Type	K	♯ inst	Gap (%)	♯ opt	♯ ub	Gap (%)	♯ opt	♯ ub
R	2	30	0.02	28	30	0.01	29	30
	3	30	0.95	16	30	0.23	24	30
	4	30	3.21	3	24	1.48	19	30
	5	30	5.85	3	16	2.98	11	27
	6	30	8.51	3	8	5.09	10	25
E	2	15	0.09	13	15	0.09	13	15
	3	15	1.67	5	15	0.45	11	15
	4	15	4.02	1	13	1.94	9	15
	5	15	6.59	1	6	3.71	6	12
	6	15	8.11	1	5	4.94	5	12
Total		225		**74**	162		**137**	211

Table 5.3: Comparison of branch–and–cut procedures on the LC K–RPP(0) instances

	Gap0 (%)	Gap (%)	Time	♯ opt	♯ ub
BC2	5.48	2.03	91.5	137	211
basic B&C	10.27	9.14	799.7	45	97

Table 5.4: Impact of the proposed valid inequalities in the performance of the B&C

the gap calculated with the final lower bound. Column “Time” shows the average time in seconds used by each algorithm in those instances solved to optimality by both of them, while columns 4 and 5 report the number of optimal solutions obtained and the number of instances in which at least one feasible solution has been found, respectively. As can be seen, the procedure using all the inequalities clearly outperforms the basic B&C in all the reported measures, which shows the impact of using these inequalities for solving the problem.

Chapter 6

The multi–purpose K–drones general routing problem

Nowadays, commercial drones are deployed for a wide variety of missions, with two broad categories being *delivery* and *sensing*. Delivery drones can be used to deliver items to discrete locations, such as packages, medical items, and disaster relief supplies. Another type of drone delivery in agricultural applications involves delivery of liquid chemicals (e.g., herbicides, fertilizers) or seeds and seedlings to a field or region (see DJI Agriculture, 2022). This type of agricultural delivery operation has more in common with agricultural drone sensing operations, as in both a continuous region of land is involved (e.g., a farm field) rather than discrete points for package delivery. Drones for sensing operations such as mapping, surveillance, inspection, and search activities, are equipped with environmental sensors (optical cameras, multispectral cameras, thermal sensors, radars of various types, magnetometers) to collect data about ground, underground or infrastructure conditions (plant health, buried objects, wildfires, flooding, road traffic, bridge conditions). These sensing operations are accomplished by flying the drone over a region of land or an item of infrastructure in a path that provides coverage of the area of interest. The region to be covered may be a 2– or 3–dimensional surface (e.g., a bounded terrain area or a bridge) or a network defined by infrastructure (e.g., roads, power transmission lines, or pipelines). There are several benefits for drone sensing and delivery missions compared to "manned" operations, including cost reduction and time savings, ability to access to difficult–to–reach areas, and ability to operate in settings that are too dangerous for a person, or that require a level of accuracy that only technology achieves.

Drone flights have typically been designed for a single purpose (e.g., either mapping or delivery), although some drones can be reconfigured to accomplish different purposes on different trips (Aerolab, 2022; HD Air Studio, 2022). It is important to distinguish between a drone that can be (re)configured to perform different tasks on separate trips, and a drone that can perform multiple tasks on the same trip. In this chapter, "multi–purpose drones" refers to the latter case. In a typical drone delivery, the drone makes an empty return trip after the last delivery (assuming all deliveries are successful) and this deadheading seems inherently wasteful. If such a drone could be used for other activities such as mapping or sensing, then the return trips would be less wasteful. Similarly, if a drone being used for sensing was able to make a delivery close to the flight path needed

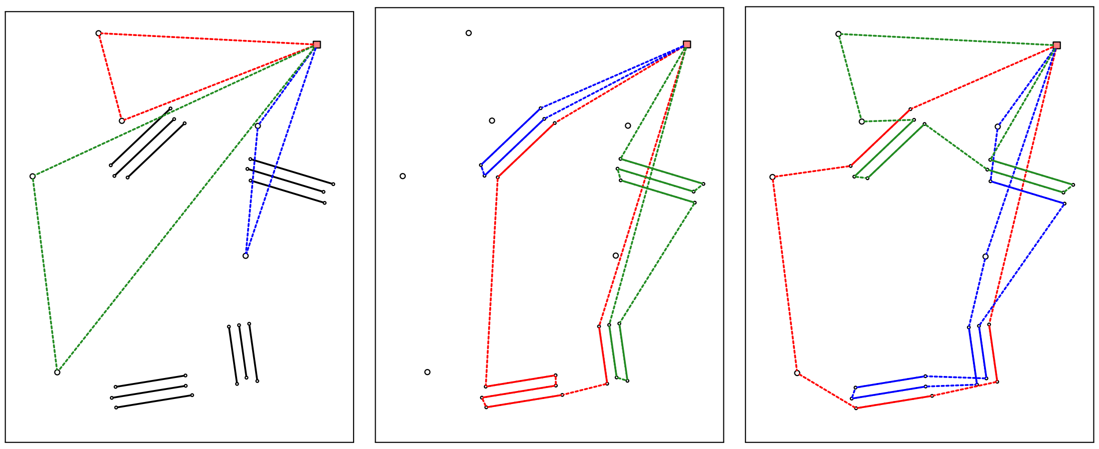

(a) Delivery routes (b) Imaging routes (c) Multi–purpose routes

Figure 6.1: Single–purpose drones VS multi–purpose drones

for sensing, then efficiency might be gained. Optimizing the routing of drones for flights that combine delivery and sensing in one trip is the key issue addressed here.

We present and study in this chapter the Multi–Purpose K–Drones General Routing Problem (MP K–DGRP). This problem was originally motivated by a UNICEF project in Malawi for multi–purpose drones that could both make deliveries of healthcare items and perform imaging as well (UNICEF, 2019; UNICEF, 2021). In the MP K–DGRP, K drone routes are to be determined to both provide sensing over a set of regions (representing flooded areas, areas with a disease outbreak or any other type of continuous areas), and also make deliveries to one or more discrete locations. Sensing by a single drone covers a linear swath as the drone flies with the width of the swath (and the resolution) determined by the sensor and the altitude of the drone. To cover two–dimensional surfaces, we can replace each region of interest by a set of parallel lines to be flown by the drones to provide a complete coverage over it. Typically, the parallel lines are oriented to minimize the number of turns by the drone, as turns interrupt the data collection and increase the drone flight time and the battery usage (see Torres et al., 2016). Since we allow a region to be covered on several different drone trips, the optimal orientation of the flights might be different for different parts of the region. We adopt here the reasonable strategy for covering a region in which drones fly in parallel lines oriented with the longest axis of the region (thus minimizing the turns needed by the drones), where the spacing of the lines reflects the characteristics of the sensor being used by the drones. Although finding the optimal way of designing these parallel lines is an interesting and difficult problem itself, in this work we will assume that the lines that drones have to follow in order to provide sensing over the areas are parallel, straight and already given. Thus, given a set of lines covering the areas to be mapped and a set of points with a certain demand, the MP K–DGRP consists of designing drone routes of lowest total duration, jointly traversing all the lines and visiting all the points (to make deliveries). The duration of each route cannot exceed a time limit and the demand delivered by each route cannot exceed the capacity of the drone. As in the LC K–DRPP addressed in previous chapters, drones can travel off the network, entering and leaving each given line at any point of it, and therefore the MP K–DGRP is a continuous optimization problem.

In our problem, rather than have one trip for deliveries and a second trip for imaging (using a reconfigured or different drone), a single drone trip might perform both deliveries and imaging. Using multi–purpose drones instead of single–purpose ones can lead to considerable improvements in the total duration of the trips, as can be seen in the example illustrated in Figure 6.1. This example includes six delivery points and four sets of parallel lines that are designed to provide mapping coverage of four regions. Figure 6.1a shows the optimal routes performed by three drone trips delivering packages to the six customers, with a total duration of 3234.65, while Figure 6.1b presents the optimal routes of three drone trips covering four areas to be mapped, whose total duration is 4361.63. However, using three multi–purpose drone trips, as can be seen in Figure 6.1c, the delivery and imaging can be carried out with a total duration of 5574.07, which is 26.6% lower than for the six drone trips using single–purpose drones.

The contributions of this chapter are arranged as follows. Section 6.1 reviews some related works concerning coverage of regions and multi–purpose drones applications. In Section 6.2, we formally describe the MP K–DGRP, and also define and propose a formulation for its discrete version, the K–vehicles general routing problem (K–GRP), and present several valid inequalities for it. In Section 6.3, a branch–and–cut algorithm for the solution of the K–GRP is proposed, while Section 6.4 is devoted to the description of a matheuristic for the MP K–DGRP. The computational experiments carried out with both algorithms are presented in Section 6.5.

6.1 Literature review

There is very limited research on multi–purpose drone problems that combine delivery with mapping or other activities in the same trip by the same drone. In Khosravi et al. (2021), the authors study a problem in which the aim is to design drone trajectories that perform some transportation operation, such as package delivery, while also providing uniform coverage (over time) of a neighborhood area. They investigate the use of multi–purpose drones in a simplified scenario where the neighborhood area is a circular region, and in another scenario where the area is an arbitrarily shaped region. For both scenarios, the authors propose an algorithm for uniform coverage and last–mile delivery applications and show that both algorithms provide a uniform coverage probability for a typical user within the neighborhood area. McCormack and Stimberis (2010) and Silvagni et al. (2017) address avalanche scenarios. The first authors present the idea and some evidence of using drones to detect an area likely to be avalanched over a road, and then for accurately dropping explosive charges to trigger controlled avalanches. The work in Silvagni et al. (2017) is a bit different in using the drone to detect (thermally) a buried body and then drop an avalanche beacon nearby. Both articles involve a single drone trip making a delivery after some sensing.

Other articles have studied the routing of drones to monitor (cover) one or several areas. Xie et al. (2020) study a path planning problem consisting of finding the optimal tour for a single drone that has to cover multiple separated convex polygonal regions. In this paper, the authors provide a mixed integer programming formulation and propose a procedure for covering a single convex polygonal region. Based on this method, they develop two approaches to solve the complete problem, a dynamic programming–based exact algorithm and a heuristic to generate high–quality tours. Puerto and Valverde

(2021) address the problem of designing routes for drones that must visit a number of geographical elements to deliver some good or service. They present two formulations that are tested on a set of instances with different shapes of elements, second order cone representable and polyhedral neighborhoods and polygonal chains. Yang et al. (2018) studied the construction of a route for a drone that has to map an area while minimizing the total distance. The authors split the area into squares and present an ant colony metaheuristic to design the route that visits them. Wang et al. (2018) consider the routing of drones that must "supervise" disjoint areas over a given time horizon. The areas are divided into cells which must be visited several times within a time period. A multiobjective evolutionary algorithm to solve the problem is proposed. Vasquez–Gomez et al. (2018) propose a method for finding a path for a drone covering several disjoint areas consisting of two steps: first determining the visiting order of the areas and then the optimization of the flight lines orientation. In Vasquez–Gomez et al. (2020), the authors address the same problem with only one region but taking into account the starting and ending points of the drone.

There are some reported applications of multi–purpose drone trips in global healthcare settings. Drone provider Swoop Aero, in association with the UK Department for International Development and UNICEF Malawi, studied the benefits of using multipurpose drones in Malawi to improve access to healthcare for remote communities and improve disaster preparedness through aerial mapping (see Swoop Aero [151]). In 2020, they undertook a 10–month sustained multi–purpose air medical logistics and disaster relief operation in two districts of Malawi to combine daily long–range drone deliveries with flood mapping and disaster response. A report on this project says that "Swoop Aero successfully proved the multifunctionality of the technology–based platform to conduct simultaneous aerial mapping tasks as well as routine medical commodity deliveries within the south of the country" (see Swoop Aero [152]). Another example of multi–purpose drone trips in global health is Deloitte [153], which notes that a drone delivering medical supplies could also be equipped with a sensor to obtain agricultural data from farms the drone passes on the way, serving two different users at once. Another project in Malawi is described in [154] in which a single UAV performs three separate public services: medical supply, soil mapping for UNICEF (inspection and imaging that could be used for agriculture, infrastructure and development projects), and aerial surveillance for the Malawi Ranger Service to monitor endangered species and detect poaching. A very different example of multipurpose drone trips that includes sensing and collection (instead of delivery) is Galle et al. (2021), which describes the use of a drone for sensing near erupting volcanes. Here, the drone has sensors to measure concentrations of various molecules and environmental conditions in the volcanic plume, and it also collects physical samples of the volcanic plume.

There are also applications for multi–purpose drones that focus on communication of the data acquired in the sensing operations. In these cases, the delivery component is downloading (or uploading) information, which is accomplished by having the drone travel to close proximity to a base station or other drone. Kumar et al. (2020) consider a system with one drone route that conducts video surveillance over a rectangular region using repeated parallel passes of the drone, followed by travel to a delivery location to communicate the information collected to a base station. This research focuses on specifics of the communication channels and communication protocols, rather than the drone routing. Cuong et al. (2022) consider a similar problem with surveillance using a camera of a "long strip" as for linear infrastructure (e.g., power transmission line, roadway or pipeline), followed by the delivery of the captured images to a base station. This includes problems where a single drone can surveille the width of the strip in one pass, and an extension where multiple drones fly together down the length of the strip (as each drone's camera can cover only part of the width of the strip). Routing for the sensing component is simplified by following one linear feature, and the research focuses on the data transmission to the base station node with data transmission rate constraints. Combining sensing and delivery of information (communication) to a base station is a growing area of research as "there is huge interest to perform joint communications and sensing now" (New and Leow, 2021).

6.2 The multi–purpose K–drones general routing problem

Let us consider a finite family $\mathcal{A}$ of separated continuous areas, a set $\mathcal{V}$ of isolated locations and a fleet of multipurpose drones located at a depot v_0 (see top image in Fig. 6.2). Multipurpose drones have both imaging and delivery capability, and they have to provide sensing over the continuous regions in $\mathcal{A}$ and also make a certain delivery at each location in $\mathcal{V}$, without running out of battery and without exceeding their load capacity. We assume that each region or area $A \in \mathcal{A}$ is described (covered) by a set of virtual parallel straight lines so that, by traversing these lines, drones cover the whole area (see bottom image in Fig. 6.2). The separation distance between the parallel lines depends on technical aspects of the drone and its sensing equipment (such as the size of the camera footprint, the altitude of the flight of the drones, etc.). Moreover, drones are capable of delivering goods to the locations of $\mathcal{V}$, assuming that the load they can transport is limited and the demand of each location must be delivered by the same vehicle at one time.

The MP K–DGRP can be defined as follows. Let us consider a set of lines, each one with an associated service (traversing) time, a depot v_0, and a set of locations with an associated positive demand and a service (visiting) time, and assume that the time of deadheading between any two points is the Euclidean distance. Given two constants L and $\mathcal{Q}$, the problem consists of finding a set of drone routes starting and ending at the depot, with duration no greater than L and load not exceeding $\mathcal{Q}$, such that they jointly traverse all the given lines and satisfy the demand of all the locations with minimum total duration. In this problem, we consider that a drone can travel in a straight line between any two points, and not necessarily following the links of the given network as happens in classical routing problems with ground vehicles. Thus, a drone can enter a line that requires service through any of its points, traverse and service part of it, exit the line through another of its points, then travel directly to another line or to make a delivery at a required location, and continue its route while the time limit L is not exceeded.

The ability of drones to enter or leave each line at any of its points allows better solutions (lower cost routes) than those obtained with traditional vehicles that cannot travel off of a network, but it makes the optimization problem continuous and very difficult to address. To deal with this issue, as was done for the LC K–DRPP in the previous chapters, we select for each instance a finite number of points on each line, so that drones can enter and leave each line only at these points. In other words, each

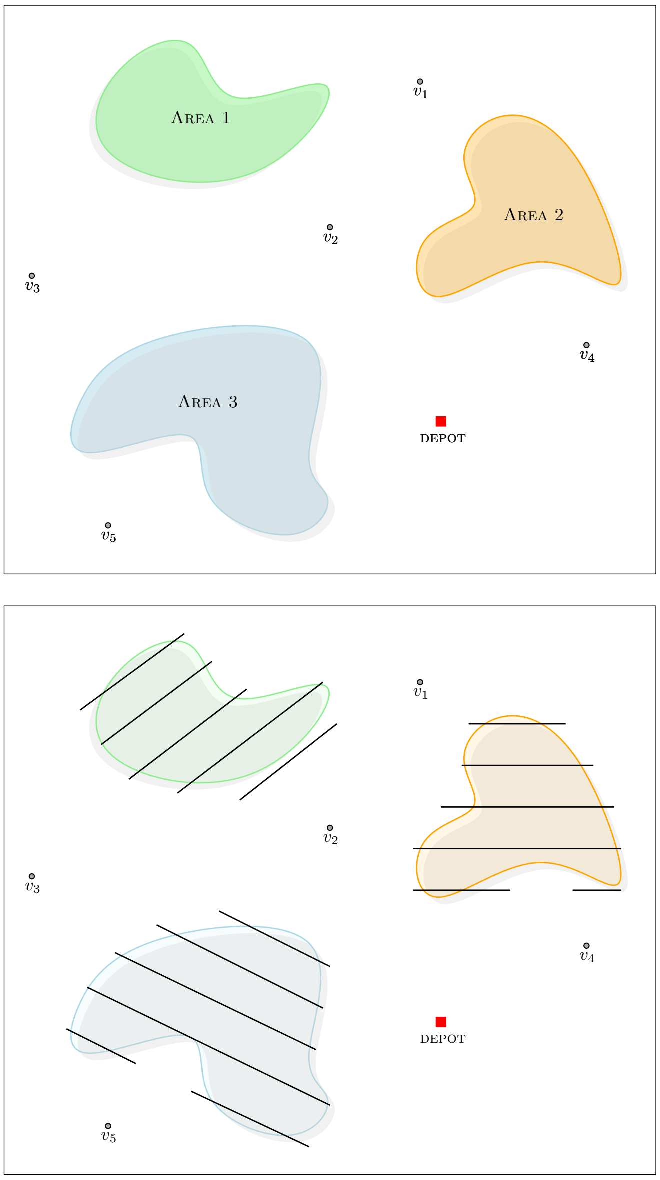


Figure 6.2: A multi–purpose drone framework with $|\mathcal{A}| = 3$ and $|\mathcal{V}| = 5$

original line is divided into small edges that must be traversed by the drones. In this way, for example, a single edge with two endpoints may have four added intermediate points that transform the one original edge into five edges, each with time of traversal equal to the proportional part of the time of traversal of the corresponding original edge. Thus, the continuous problem is reduced to a discrete problem, the K–VEHICLES GENERAL ROUTING PROBLEM (K–GRP).

Obviously, the more intermediate points selected on each line, the better is the approximation of the original continuous problem, and thus better solutions might be obtained. However, if the number of intermediate points is very large, the size of the instance increases so much that it cannot be addressed, and even heuristic algorithms can fail to provide good solutions in reasonable computing time. Thus, it is necessary to devise sophisticated strategies to generate instances of the K–GRP with a reduced but significant number of intermediate points that can be solved to produce good solutions for the MP K–DGRP.

In what follows, we formally define the K–GRP and propose a formulation and some valid inequalities for it.

6.2.1 A formulation for the K–vehicles general routing problem

Given an undirected and connected graph $G = (V, E)$, a subset $E_R \subseteq E$ of required edges and a subset $V_R \subseteq V$ of required vertices, the general routing problem (GRP; Orloff, 1974) consists of finding a minimum cost tour (closed walk) on G traversing each edge $e \in E_R$ and visiting each vertex $v \in V_R$ at least once.

In the K–GRP, the required edges $E_R \subseteq E$ correspond to the segments into which we have split the straight lines that have to be traversed in order to completely cover the continuous areas. Since these lines are isolated straight lines, that is, there are no lines adjacent to each other, when they are discretized we obtain isolated chains of required edges. Therefore, the vertices incident with edges in E_R are incident with either one required edge (we call this set of vertices V_O) or with two required edges (we call this set of vertices V_E). We denote by E_{NR} the set of non–required edges, which form a complete graph with the node set V.

There is a time $c_e^s \geq 0$ associated with the traversal and service of each required edge $e \in E_R$, and a deadheading time $c_e \geq 0$ associated with the traversal of each non–required edge $e \in E_{NR}$. The deadheading times satisfy the triangular inequality. The set of required vertices $V_R \subseteq V$ is formed by all the nodes in V where a drone should make a delivery. We will assume that these vertices are only incident with non–required edges and have to be serviced by exactly one drone. Otherwise, a simple transformation can be applied to meet this condition. Each node $i \in V_R$ has an associated positive integer demand $d_i > 0$ and a service time $c_i \geq 0$. Furthermore, the drone routes start and end at a vertex (the depot, vertex v_0) that, for the sake of simplicity, we will assume is not incident with required edges. Hence, we have $V = \{v_0\} \cup V_R \cup V_O \cup V_E$.

There is a fleet of K drones, each with a load capacity $\mathcal{Q}$ and a time limit L. The objective of the K–GRP is to find K routes with minimum total duration, starting and ending at the depot, that jointly traverse all the required edges and visit the required vertices exactly once, so that the duration and the load of each route do not exceed the values L and $\mathcal{Q}$, respectively. We will call K–GRP *solution* to any set of K tours that meet all the constraints of the problem.

The K–GRP can be formulated using a binary variable x_e^k for each edge $e \in E_R$ and for each drone $k \in \{1, \ldots K\}$, and two binary variables x_e^k and y_e^k for each edge $e \in E_{NR}$ and for each drone k. Variable x_e^k takes the value 1 if the required edge e is traversed (and serviced) by drone k and 0 otherwise, while variables x_e^k and y_e^k take the value 1 if the non–required edge e is traversed or traversed twice by drone k, respectively, and 0 otherwise. In other words, variables x_e^k and y_e^k represent the first and second traversal of non–required edge e by drone k. Additionally, a binary variable z_i^k is introduced for each vertex $i \in V_R$ and each drone k, taking the value 1 if the vertex i is serviced by drone k and 0 otherwise.

Again, we 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 $\delta(S) = (S : V \backslash S)$ 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 $F_R = F \cap E_R$, $F_{NR} = F \cap E_{NR}$, $x^k(F) = \sum_{e\in F} x_e^k$ and $(x^k + y^k)(F) = \sum_{e=(i,j)\in F}(x_e^k + y_e^k)$.

We propose the following formulation for the K–GRP:

$$\text{Minimize} \qquad \sum_{k=1}^{K} \sum_{e\in E_{NR}} c_e\big(x_e^k + y_e^k\big) + \sum_{k=1}^{K} \sum_{e\in E_R} c_e^s x_e^k + \sum_{k=1}^{K} \sum_{i\in V_R} c_i z_i^k \tag{6.1}$$

s.t.

$$\sum_{e\in\delta_R(i)} x_e^k + \sum_{e\in\delta_{NR}(i)} \big(x_e^k + y_e^k\big) \equiv 0 \pmod 2, \qquad \forall i \in V\backslash V_R,\ \forall k \tag{6.2}$$

$$\sum_{e\in\delta(i)} (x_e^k + y_e^k) = 2\, z_i^k, \qquad \forall i \in V_R,\ \forall k \tag{6.3}$$

$$\sum_{k=1}^{K} x_e^k = 1, \qquad \forall e \in E_R \tag{6.4}$$

$$\sum_{k=1}^{K} z_i^k = 1, \qquad \forall i \in V_R \tag{6.5}$$

$$x_e^k \geq y_e^k, \qquad \forall e \in E_{NR},\ \forall k \tag{6.6}$$

$$\sum_{e\in\delta_R(S)} x_e^k + \sum_{e\in\delta_{NR}(S)} \big(x_e^k + y_e^k\big) \geq 2x_f^k, \qquad \forall S \subseteq V\backslash\{v_0\}, \forall f \in E(S), \forall k \tag{6.7}$$

$$\sum_{e\in E_R} c_e^s x_e^k + \sum_{e\in E_{NR}} c_e\big(x_e^k + y_e^k\big) + \sum_{i\in V_R} c_i z_i^k \leq L, \qquad \forall k \tag{6.8}$$

$$\sum_{i\in V_R} d_i z_i^k \leq \mathcal{Q}, \qquad \forall k \tag{6.9}$$

$$z_i^k \in \{0,1\}, \qquad \forall i \in V_R,\ \forall k \tag{6.10}$$

$$x_e^k \in \{0,1\}, \qquad \forall e \in E_R,\ \forall k \tag{6.11}$$

$$x_e^k, y_e^k \in \{0,1\}, \qquad \forall e \in E_{NR},\ \forall k \tag{6.12}$$

The objective function (6.1) minimizes the total duration of the routes. The first term represents the deadheading time from traveling on non–required edges. The second and third terms represent, respectively, the time of servicing the required edges and the time of visiting the required nodes (for delivery). Due to constraints (6.4) and (6.5), these last two terms are constant and could be removed from the objective function. Constraints (6.2) force that the number of times a drone visits a non–required vertex is even, possibly zero, and equalities (6.3) ensure that a drone traverses two non–required edges incident with a required vertex i if it is serviced by this drone (and traverses no incident edges if it is not serviced). Equations (6.4) force each required edge to be serviced exactly once, and the visit of each required vertex by exactly one drone is ensured by constraints (6.5). Constraints (6.6) guarantee that a second traversal of a non–required edge by a drone can only occur when it has been traversed previously by this drone. Inequalities (6.7) avoid subtours and ensure that each single route is connected and connected to the depot. Constraints (6.8) guarantee that the duration of each route does not exceed L, while constraints (6.9) ensure that the demand serviced on each route is not greater than the drone capacity $\mathcal{Q}$. Constraints (6.10), (6.11) and (6.12) are the binary conditions for the variables.

The following results allow us to remove some variables from the formulation, taking into account the structure of the optimal K–GRP solutions.

Theorem 6.2.1. *There is an optimal K–*GRP *solution in which each route satisfies:*

(a) *It does not deadhead two edges (i, j), (j, k) consecutively, except if $j \in V_R$ and it is serviced by the route.*

(b) *It does not visit the vertices in $\{v_0\} \cup V_R \cup V_O$ more than once.*

(c) *It does not visit the vertices in V_E more than twice.*

Proof: (a) Let us suppose a tour deadheads two edges (i, j), (j, k) consecutively with $j \notin V_R$. Each one of these two edges is either a non–required one or a required edge that had already been serviced in a previous traversal. Recall that the non–required edge (i, k) exists because E_{NR} induces a complete graph, and that $c_{ik} \leq c_{ij} + c_{jk}$ due to the triangle inequality. The tour obtained after replacing the traversal of $(i, j), (j, k)$ with the traversal of (i, k) has a lower or equal duration and services the same required edges and the same required vertices as the former tour. By iterating this argument we obtain (a).

(b) The drone tours start and end at v_0. If a tour visits v_0 again, it deadheads two edges without servicing v_0, which contradicts (a). Let us suppose a tour visits a given vertex $j \in V_R \cup V_O$ more than once. Note that there is only one service associated with vertex j: the service of j, if $j \in V_R$ or the service of the unique required edge incident with j, if $j \in V_O$. Therefore, all visits to the vertex j, except one of them, are performed by deadheading two edges without servicing j, which contradicts (a).

(c) Let us suppose a tour visits a given vertex $j \in V_E$ more than twice. Note that there are only two services associated with vertex j: the service of the two required edges incident with $j \in V_E$. Therefore, all visits to the vertex j, except two of them, are performed by deadheading two edges without servicing j, which contradicts (a). $\square$

Corollary 6.2.2. *For each drone k, variables y_e^k for the following non–required edges e can be removed from the formulation:*

- *$e = (u, v)$ with $u \in V_O$ (or $v \in V_O$), and*
- *$e = (u, v)$ with $u, v \in V_R$.*

and the following inequalities can be added to the formulation:

$$x^k(\delta(i)) = 2x_{ij}^k, \qquad \forall (i, j) \in E_R, \text{ with } i \in V_O, \forall k \tag{6.13}$$

$$x^k(\delta(i)) \leq 2x_{ij}^k + 2x_{li}^k, \qquad \forall (i, j), (l, i) \in E_R \ (i \in V_E), \ \forall k \tag{6.14}$$

Note: Consider the special case of the K–GRP in which the lines are not split, that is, $V_E = \emptyset$. From Corollary 6.2.2, this case can be formulated using variables y_e^k only for those edges joining the depot and vertices in V_R (all the other y_e^k variables can be removed). Moreover, from Theorem 6.2.1, there is an optimal solution that does not use any non–required edge parallel to a required one (otherwise, two consecutive edges would be deadheaded), so no variable x_e^k is needed for these edges.

6.2.2 Other valid inequalities for the K–GRP

In this section we present four families of valid inequalities that reinforce the above formulation. They are based on the capacity and autonomy constraints of drones or on known families of valid inequalities proposed in Corberán et al. (2013) for the maximum benefit Chinese postman problem (MBCPP).

Parity inequalities

Parity inequalities are obtained from those proposed in Corberán et al. (2013) for the MBCPP, which generalize the well known co–circuit inequalities (Barahona and Grötschel, 1986). They rely on the fact that each drone tour traverses any edge cut–set an even (or zero) number of times:

$$x^k(\delta_R(S)\backslash F_R) + (x^k - y^k)(\delta_{NR}(S)\backslash F_{NR}) \geq x^k(F_R) + (x^k - y^k)(F_{NR}) - |F| + 1, \quad (6.15)$$

for each drone $k \in \{1, \dots K\}$, for all $S \subset V$, and for all $F \subseteq \delta(S)$ with $|F|$ odd.

In order to see that parity inequalities (6.15) are valid for the K–GRP on G, note that $x^k(F_R)+(x^k-y^k)(F_{NR}) \leq |F|$. For a drone k, the tours that satisfy $x^k(F_R)+(x^k - y^k)(F_{NR}) = |F|$ traverse each edge in F once, and, since $|F|$ is odd, they traverse at least one edge in $\delta(S)\backslash F$ once, thus satisfying $x^k(\delta_R(S)\backslash F_R) + (x^k - y^k)(\delta_{NR}(S)\backslash F_{NR}) \geq 1$. For the drone tours satisfying $x^k(F_R) + (x^k - y^k)(F_{NR}) \leq |F| - 1$, inequality (6.15) reduces to $x^k(\delta_R(S)\backslash F_R) + (x^k - y^k)(\delta_{NR}(S)\backslash F_{NR}) \geq 0$, which is obviously satisfied.

p–connectivity inequalities

p–connectivity inequalities are based on those with the same name proposed for the MBCPP in Corberán et al. (2013) and are generalized here to also consider the existence of required vertices.

Let $\{S_0, \dots, S_p\}$ be a partition of V. Assume that $v_0 \in S_d$, $d \in \{0, \dots, p\}$, and select either an edge $e_j \in E_R(S_j)$ or a vertex $v_j \in S_j \cap V_R$ for each $j \in \{0, \dots, p\}\backslash\{d\}$. Let I_1 and I_2 be the sets of indices in $\{0, \dots, p\}\backslash\{d\}$ in which a required edge or a required vertex has been selected, respectively. For each drone k, the following inequality

$$x^k(\delta_R(S_0)) + (x^k + y^k)(\delta_{NR}(S_0)) + 2 \sum_{1 \leq r < t \leq p} x^k(S_r : S_t) \geq 2 \sum_{j \in I_1} x^k_{e_j} + 2 \sum_{j \in I_2} z^k_{v_j} \quad (6.16)$$

is valid for the K–GRP and will be referred to as p–connectivity inequality.

Theorem 6.2.3. *p–connectivity inequalities* (6.16) *are valid for the K–*GRP.

Proof: Let us assume that the depot belongs to S_0 and let $(\bar{x}, \bar{y}, \bar{z})$ be a K–GRP tour. Given a drone k, if there is an edge e_j, $j \in I_1$, or a vertex v_j, $j \in I_2$, such that $\bar{x}^k_{e_j} = 0$ or $\bar{z}^k_{v_j} = 0$, we can consider another p–connectivity inequality with $p-1$ subsets by merging sets S_j and S_{j+1} (or S_{j-1}). If the new $(p-1)$–connectivity inequality is satisfied by the solution, the original one will also be. Therefore, we can assume that $\bar{x}^k_{e_i} = 1$ for all $i \in I_1$ and $\bar{z}^k_{v_i} = 1$ for all $i \in I_2$.

Moreover, if $\bar{x}^k(S_r : S_t) \geq 1$, we can also merge S_r and S_t, obtaining a new $(p-1)$–connectivity inequality. As before, if this inequality is satisfied, the original one also holds. Therefore we can also assume that $\bar{x}^k(S_r : S_t) = 0$ for any pair of sets S_r, S_t, with $1 \leq r < t \leq p$.

Since each set S_i, $i > 0$, must be connected to the depot and $\bar{x}^k(S_r : S_t) = 0$, $x^k(\delta_R(S_0)) + (x^k + y^k)(\delta_{NR}(S_0)) \geq 2(|I_1| + |I_2|)$, and the inequality holds. $\square$

Capacity inequalities

Capacity inequalities are widely used in node and arc routing problems when there is a limit to the demand a vehicle can service. Given a vertex set $S \subseteq V \backslash \{v_0\}$, the edge cut–set $\delta(S)$ has to be traversed at least twice the number of vehicles needed to service the demand of the required vertices in S, and, therefore, the following inequality is valid:

$$\sum_{k=1}^{K} x^k(\delta_R(S)) + \sum_{k=1}^{K} (x^k + y^k)(\delta_{NR}(S)) \geq 2 \left\lceil \sum_{i \in V_R \cap S} d_i / Q \right\rceil, \quad \forall S \subseteq V \backslash \{v_0\}. \qquad (6.17)$$

Max–time inequalities

Max–time inequalities are based on the limit L to the duration of a drone tour. Consider two sets $F \subseteq E_R$ and $S \subseteq V_R$ such that the duration of the optimal tour (or a lower bound to it) starting and ending at v_0, traversing all the edges in F, and visiting all the vertices in S, is greater than L. On the one hand, we have that a single drone k cannot service all the arcs in F and all the vertices in S and, hence, inequalities

$$x^k(F) + z^k(S) \leq |F| + |S| - 1, \quad \forall k \in \{1, \dots K\}, \qquad (6.18)$$

are valid for the K–GRP on G. On the other hand, under the same circumstances, at least two different drones must enter any subgraph of G that contains all edges in F and all vertices in S but does not contain the depot. Hence, if $W \subset V \backslash \{v_0\}$ is a set of vertices containing S and the vertices incident with the edges in F, the following inequalities

$$\sum_{k=1}^{K} \Big(x^k(\delta_R(W)) + (x^k + y^k)(\delta_{NR}(W)) \Big) \geq 4, \qquad (6.19)$$

and

$$\sum_{\substack{k'=1 \\ (k' \neq k)}}^{K} \Big(x^{k'}(\delta_R(W)) + (x^{k'} + y^{k'})(\delta_{NR}(W)) \Big) \geq 2, \quad \forall k \in \{1, \dots K\} \qquad (6.20)$$

are also valid for the K–GRP on G. Inequalities (6.18), (6.19), and (6.20) can be easily generalized to the case when the number of drones needed to service the edges in a given set F and the vertices in a given set S is greater than two.

6.3 A branch–and–cut algorithm for the K–GRP

We have implemented a branch–and–cut algorithm for the K–GRP based on the formulation proposed in Section 6.2.1. The fundamental component of the branch and cut is a cutting–plane algorithm that incorporates separation procedures for the inequalities presented in Section 6.2.2.

6.3.1 Separation algorithms

At each iteration of the cutting–plane procedure we use some separation algorithms to identify valid inequalities that are violated by the current LP fractional solution. Let $(\bar{x}^k, \bar{y}^k, \bar{z}^k)$, for $k = 1, \ldots, K$, denote this fractional solution, and let $\varepsilon > 0$ be a given parameter. In what follows we describe the separation procedures used for each family of valid inequalities.

Connectivity inequalities

Although connectivity inequalities (6.7) can be exactly separated in polynomial time by means of max–flow computations, the procedure is very time consuming and may produce many similar violated inequalities. Therefore, we decided to use heuristic algorithms for separating them. The first one is a well–known method based on the computation of the connected components of the graph induced by the edges $e \in E$ such that $\bar{x}^k(e) + \bar{y}^k(e) > \varepsilon$, plus the depot, if necessary. Inequality (6.7), associated with each connected component and with the edge f in it with maximum $\bar{x}^k(f)$, is checked for violation.

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

Parity inequalities

Parity inequalities can be exactly separated in polynomial time with an algorithm based on the Padberg–Rao procedure (Padberg and Rao, 1982) for the computation of odd minimum cuts (see also Letchford et al. (2008) for a more efficient procedure). For the special case where $S = \{v\}$, the exact procedure described in Ghiani and Laporte (2000) gives the set $F \subseteq \delta(v)$ associated with the most violated parity inequality.

We also use a simple and fast algorithm based on the computation of the connected components of the graph induced by the edges $e \in E$ with $\bar{x}^k(e) - \bar{y}^k(e) > \varepsilon$, plus the depot, if necessary. The same procedure as for the case $S = \{v\}$ is used here to obtain the set F corresponding to the cut–set associated with each component.

p–connectivity inequalities

To separate these inequalities, we first look for tight connectivity inequalities (6.7) by searching among the cut–sets obtained with the connectivity separation procedures. Let S be a set of vertices associated with such a cut–set and assume that $v_0 \in S_0 = V \setminus S$. If the connectivity inequality is tight for drone k, we compute the connected components in the subgraph induced in $G(S)$ by the edges $e \in E(S)$ with $\bar{x}^k(e) \geq 1 - \varepsilon$. For each pair C_i, C_j of such components,

$$s_{ij} = 2\bar{x}^k(V_i, V_j) - 2\min\{\bar{w}^k(\alpha_i), \bar{w}^k(\alpha_j)\}$$

is calculated, where V_i and V_j denotes the set of vertices in C_i and C_j, and

$$\bar{w}^k(\alpha_t) = \max\{\bar{x}^k(e) : e \in E(V_t), \bar{z}^k(v) : v \in V_t \cap V_R\},$$

for each component 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 associated inequality (6.16) is checked for violation.

Capacity inequalities

Capacity inequalities (6.17) 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 non–required edges with

$$\bar{c}_e = \sum_{k=1}^{K} (\bar{x}_e^k + \bar{y}_e^k) > 0.$$

Required edges have unit capacity and edges in $\overline{E}_{NR}$ have capacity $\bar{c}_e$. For each required vertex $i \in V_R$, we compute the max flow in $\overline{G}$ from the depot to i. Let $(S : V \setminus S)$ be the associated minimum cut, with $i \in S$, and

$$n_v = \left\lceil \sum_{j \in V_R \cap S} d_j / Q \right\rceil.$$

If the max flow is less than $2n_v$, this cutset defines a violated capacity inequality (6.17). Whether the inequality is violated or not, we define $\mathcal{K}$ as the set of vehicles k such that $\bar{x}^k(\delta_R(S)) + (\bar{x}^k + \bar{y}^k)(\delta_{NR}(S)) < 2$. Then, the inequality

$$\sum_{k \in \mathcal{K}} \bar{x}^k(\delta_R(S)) + \sum_{k \in \mathcal{K}} (\bar{x}^k + \bar{y}^k)(\delta_{NR}(S)) \geq 2(n_v - K + |\mathcal{K}|), \tag{6.21}$$

for all $S \subseteq V \setminus \{v_0\}$, is valid for the K–GRP and can be more violated than the initial one.

Max–time inequalities

We separate max–time inequalities (6.18), (6.19), and (6.20) by using two heuristic procedures based on the ones presented in Section 5.1.1 for the LC K–RPP.

The first one looks for violated inequalities (6.18). Let $\{e_1, e_2, \ldots, e_m\}$ be a set of required edges such that $\bar{x}^k_{e_1} \geq \bar{x}^k_{e_2} \geq \ldots \geq \bar{x}^k_{e_m} \geq 0.5$, and let $F = \{e_1, e_2, \ldots, e_f\}$, where f is the maximal number such that $\bar{x}^k(F) > |F| - 0.5$. Then we define S as the set of vertices with $\bar{z}^k_i = 1, i \in V_R$. Now we solve the GRP with the set of required edges F and the set of required vertices $S \cup \{v_0\}$ with the branch–and–cut algorithm described in Corberán et al. (2007). Let $C(F, S)$ be its optimal value or a lower bound. If $C(F, S) > L$, the corresponding inequality (6.18) is violated. Otherwise, for each edge $\overline{e} \in \{e_{f+1}, \ldots, e_m\}$, we consider the set $\overline{F} = F \cup \{\overline{e}\}$ and check if $C(\overline{F}, S)$ is greater than L. For each pair (F, S) (or $(\overline{F}, S)$) whose corresponding inequality (6.18) is violated, we look for the cutset of minimum weight between the depot and the edges in F and vertices in S in the support graph $\overline{G} = (V, E_R \cup \overline{E}_{NR})$ defined above for the separation of capacity inequalities. For this cutset, we check the corresponding max–time inequalities (6.19) and (6.20) for violation.

The second heuristic looks for inequalities (6.19). The procedure starts by defining $S = \{i\}$, where i is the vertex farthest from the depot such that the maximum flow from v_0 to i in $\overline{G}$ is less than $2K$. Then we iteratively add vertices to S in such a way that $\sum_{k=1}^{K} \bar{x}^k(\delta_R(S)) + \sum_{k=1}^{K}(\bar{x}^k + \bar{y}^k)(\delta_{NR}(S))$ is minimum. For each S, we compute the minimum number of vehicles needed to service all the edges in $E_R(S)$ and all the required vertices in S by solving the associated GRP. The corresponding inequality (6.19) is checked for violation. If a violated constraint (6.19) is found, at least one of the inequalities (6.20) is also violated and it is added.

6.3.2 The cutting–plane algorithm

The initial LP relaxation contains inequalities (6.3), (6.4), (6.5), (6.6), (6.8), (6.9), and the bounds on the variables. Moreover, as is usual in routing problems with several vehicles, some symmetry–breaking inequalities are added to avoid equivalent solutions.

At each iteration, the cutting–plane algorithm applies 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. If this heuristic fails in finding violated cuts, the second connectivity heuristic is applied for $\varepsilon = 0, 0.1, 0.2, 0.3, 0.4$. All the tight cut–sets obtained with the connectivity separation procedures are stored and used by the p–connectivity heuristic to check the violation of their associated inequalities with $\varepsilon = 0, 0.15, 0.3$.

Moreover, at each iteration we apply the heuristic for identifying violated capacity inequalities, the exact parity separation procedure for single vertices, and the heuristic procedure for parity inequalities with values $0, 0.25$, and 0.5 for ε. A value of ε is used only if the previous one results in no violated inequalities.

Max–time separation algorithms are called only at the root node. Furthermore, the exact procedure for parity inequalities is applied only at the root node if no violated connectivity, p–connectivity, nor parity inequalities have been found so far.

6.4 A matheuristic for the multi–purpose K–drones GRP

In this section, we present a matheuristic algorithm for the MP K–DGRP that consists of two parts. The first part aims to find solutions for the K–GRP considering that each original line of the MP K–DGRP instance is represented by a single required edge (without intermediate points). Let us call this instance K–GRP(0). In this part, we use an order–first split–second method (see Prins et al., 2014) that initially generates a "giant tour" traversing all the required edges and visiting all the required nodes of G without considering the max–time and capacity constraints, and then K feasible drone routes for the K–GRP(0) instance are obtained from it. This is described in Section 6.4.1. Several different giant tours are generated and partitioned into K routes to obtain a set of K–GRP(0) feasible solutions. These initial solutions are improved by applying a variable neighborhood descent (VND) algorithm, described in Section 6.4.2, that combines four local search procedures and a route optimization phase.

The second part of the matheuristic is focused on improving the n best K–GRP(0) solutions obtained in the previous part by adding some intermediate points to the required edges, thus allowing drones to enter (or to exit) the lines not only at its endpoints, but also at a subset of intermediate points. First we consider the new instance K–GRP(1) resulting from adding one intermediate vertex to each required edge, so that each line is approximated by a polygonal chain with two segments (edges) with the same traversal (and service) time. Each one of the n K–GRP(0) solutions previously obtained is trivially transformed into a K–GRP(1) solution. We call this procedure "1–splitting". Then, we apply again the VND algorithm and the route optimization procedure to each K–GRP(1) solution. Some of these solutions may have been now improved due to drones entering or leaving some required edge through its middle point. The most "promising" edges of each solution, those whose two halves are now serviced by different drones (or by the same drone but not consecutively), are split again by adding p equidistant intermediate vertices. We then try to improve the solution by using these new vertices. This is detailed in Section 6.4.3.

Throughout this section, we will use K–GRP *solution* to refer to a set $\mathcal{T} = (T_1, T_2, \dots, T_K)$ of K routes, each one starting and ending at the depot, with duration no greater than L and total demand not exceeding the drone capacity $\mathcal{Q}$, such that each required edge is traversed and each required node is visited. We will use *task* $t_i = (t_{i1}, t_{i2})$ to refer either to a required edge (t_{i1}, t_{i2}) serviced by traversing it from t_{i1} to t_{i2} or to a required vertex $t_{i1} = t_{i2}$. Then, a route associated with drone k can be represented by a sequence of tasks $T_k = \{t_i^k, \dots, t_j^k\}$, where it is assumed that the deadheading from the depot to the first task, from the end of a task to the beginning of the following one, and from the last task back to the depot, is done by traversing the corresponding non–required edge. We will denote by $d(t_\ell^k)$ the demand of task t_ℓ^k, for $t_\ell^k \in V_R \cap T_k$. A route will be *feasible* if its duration is not greater than L and if its total demand does not exceed $\mathcal{Q}$.

6.4.1 Solutions for K–GRP(0)

As mentioned above, this first part of the matheuristic focuses on finding solutions for the K–GRP(0) instance. The algorithm starts by finding an optimal tour on $G =$

Algorithm 1 Splitting procedure for a giant tour T_G

```
Require: $G, T_G$
 1: initialize $G^* = (V^*, A^*)$: $V^* \leftarrow \{0, 1, \dots, n\}$; $A^* \leftarrow \emptyset$
 2: for $i \leftarrow 1$ to $n$ do
 3:     $j \leftarrow i$
 4:     $arctime \leftarrow 0$
 5:     $demand \leftarrow d(t_j)$
 6:     while $(j \leq n)$ and $(demand \leq \mathcal{Q})$ and $(arctime \leq L)$ do
 7:         $arctime \leftarrow \gamma(i-1, j)$
 8:         if $arctime \leq L$ then
 9:             add arc $(i-1, j)$ to $A^*$ with traversal time $arctime$
10:             $j \leftarrow j + 1$
11:             $demand \leftarrow demand + d(t_j)$
12:         end if
13:     end while
14: end for
15: compute the shortest path $P$ from 0 to $n$ in $G^*$
16: transform each arc of $P$ into a sequence of ordered tasks
17: return a $K$–GRP(0) solution (a set of $K$ drone routes)
```

(V, E) traversing all the required edges and visiting all the required nodes of G. This tour, commonly called a *giant tour* in the literature, is generated by relaxing drone capacity and time limit, and solving the GRP instance optimally with the branch–and–cut algorithm described in Corberán et al. (2007). The tour returned by this procedure has an associated sequence of tasks T_G.

The giant tour is optimally partitioned into K feasible drone routes by solving a shortest path problem over an auxiliary directed graph $G^* = (V^*, A^*)$ (see Beasley, 1983; Ulusoy, 1985). This procedure was already used in Section 3.3.1 for the length constrained K–RPP(0) and has been extended here to consider required vertices as explained below.

The set V^* contains $|V_R| + |E_R| + 1$ nodes, where v_0 denotes the depot and the remaining nodes v_ℓ represent the tasks t_ℓ. The nodes are arranged from left to right following the order in which their associated tasks are performed in the giant tour. Each arc in A^* represents a feasible drone route on G. An arc from node v_i to node v_j is added if the route starting from the depot, performing tasks $t_{i+1}, \dots, t_j$ in that order, and going back to the depot, is feasible. The time associated with these arcs, denoted $\gamma(i, j)$, is the duration of the corresponding route.

In graph G^* we compute a shortest path from node v_0 to node $v_{|V_R|+|E_R|}$, whose set of arcs, as proved in Ulusoy (1985), defines a partition of the giant tour into K feasible tours that is optimal regarding the ordering of the traversal of the tasks. Algorithm 1 summarizes this procedure. However, this method does not take into account that required edges can be traversed in two possible directions. Prins et al. (2009) propose a procedure called *Split with Flips* that is a generalization of the previous method considering the two directions in which each edge can be traversed. The main difference consists of the way the times $\gamma(i, j)$ associated with the arcs in G^* are calculated.

We have adapted this algorithm to consider required vertices as follows. For each arc in $a \in A^*$ representing a route, a new auxiliary directed graph G^*_a is created in order to calculate its associated duration. Each task corresponding to a required edge in G is represented now with two arcs, one for each possible direction of traversal, with times equal to its original service time. For each task associated with a required vertex, one vertex is created with a (visiting) time equal to its original service time in G. Then an arc is added from the end vertex of each arc (or vertex) representing a task to the initial vertex of the following arc (or vertex), whose time is that of the corresponding shortest path in G. Two additional vertices representing the depot are added and connected with the first and last task of the route, respectively, with times equal to those of the corresponding shortest paths in G. In this graph, a shortest path between the two copies of the depot is calculated. This shortest path determines the optimal direction of traversal of each task and its time will be the one assigned to arc $a \in A^*$.

Figure 6.3 illustrates the graph generated for a subsequence $T = \{t_1, t_2, t_3, t_4, t_5\}$ of tasks, where arc t_ℓ represents task t_ℓ traversed in the direction given by the route and arc t_ℓ^{-1} corresponds to the opposite direction of traversal. The numbers next to the nodes and edges represent their service times. Dashed lines represent the non–required edges corresponding to the shortest paths joining the tasks. In this example, the shortest path $\{t_1, t_2, t_3^{-1}, t_4, t_5\}$ has value 52, while the directions given by the giant tour have a value of 58.

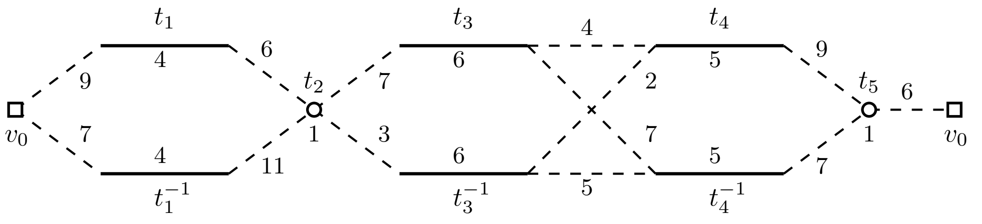


Figure 6.3: Example of an auxiliary graph used in the *Split with Flips* procedure

In order to obtain a larger set of initial solutions, we apply the above algorithm with other initial giant tours on G defined by different Eulerian circuits obtained from the optimal GRP solution. Note that a GRP 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) $|E_R| + |V_R|$ times, starting each time with a different task. We thus obtain a set $\bar{\mathcal{T}}$ of different initial K–GRP(0) solutions, with $|\bar{\mathcal{T}}| \leq |E_R| + |V_R|$.

6.4.2 A variable neighborhood descent algorithm for the K–GRP(0)

Once the initial set of K–GRP(0) feasible solutions is generated, a variable neighborhood descent algorithm (VND) (Mladenovic and Hansen, 1997; Duarte et al., 2018) is applied to each solution $\mathcal{T} \in \bar{\mathcal{T}}$ to try to improve it. A VND is a metaheuristic that explores a sequence $\mathcal{N} = (N_1, \ldots, N_{\rho_{max}})$ of neighborhood structures in a deterministic way. Starting from an initial solution $\mathcal{T}$ and $\rho = 1$, each iteration of the VND explores the neighborhood $N_\rho(\mathcal{T})$ to try finding a better solution. If one improving move is detected, it is executed and ρ is reset to 1; otherwise, ρ is incremented to browse the next neighborhood. The algorithm stops when the exploration of the last neighborhood $N_{\rho_{max}}(\mathcal{T})$ brings no improvement, that is, when the current solution $\mathcal{T}$ is a local

Algorithm 2 Pseudocode of the VND algorithm

```
1: for each $K$–GRP(0) solution $\mathcal{T} \in \bar{\mathcal{T}}$ do
2:     $\rho \leftarrow 1$
3:     while $\rho \leq 4$ do
4:         obtain $\mathcal{T}' \in \mathcal{N}_\rho(\mathcal{T})$ by applying the $\rho$–th local search method to $\mathcal{T}$
5:         if $f(\mathcal{T}') < f(\mathcal{T})$ then
6:             $\mathcal{T} \leftarrow \mathcal{T}'$ and $\rho \leftarrow 1$
7:         else
8:             $\rho \leftarrow \rho + 1$
9:         end if
10:    end while
11: end for
```

optimum over all the considered neighborhoods. The VND algorithm proposed here explores $\rho_{max} = 4$ different neighborhood structures (see Algorithm 2).

The first neighborhood structure N_1 is defined by the *intra–route* move and consists of all the permutations resulting from moving a task to another position within the route that performs it. For each route T_k of a solution $\mathcal{T}$, the intra–route procedure removes a task $t_\ell \in T_k$ at each step and inserts it in the position of T_k that minimizes the duration of the route.

The second neighborhood structure N_2 is defined by the *destroy and repair* move. Each iteration of the associated local–search procedure randomly chooses r tasks, with $2 \leq r \leq 8$, and removes them from their corresponding routes. Then, the algorithm tries to relocate each task one by one in the route and position that minimizes the total time, satisfying the time limit and the capacity constraints. 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 total time of the new solution is not better than the time of the original one, the changes made in this iteration are discarded. This procedure is repeated until one improving move is detected or until i_{max} consecutive iterations without any improvement are performed, with i_{max} a given parameter.

The third neighborhood structure N_3 is defined by the 0 *to* ℓ *– exchange* move. Each iteration of this local–search procedure removes ℓ consecutive tasks from the route servicing them and inserts all of them between two consecutive tasks of another route. The algorithm considers the removal of all the possible sets of ℓ consecutive tasks and their insertion in all the possible positions of other routes such that the duration and the total demand of the resulting route do not exceed L and $\mathcal{Q}$, respectively. The algorithm starts with $\ell = 1$, and if no exchange improves the original solution, ℓ is incremented by 1 and the process is repeated. The procedure stops when an improving move is executed, or when $\ell = \ell_{max}$ and there are no moves that improve the total time, with ℓ_{max} a given parameter.

The last neighborhood structure N_4 is defined by the ℓ_1 *to* ℓ_2 *– exchange* move and contains all the solutions obtained by interchanging a chain of ℓ_1 consecutive tasks from one route with a chain of ℓ_2 consecutive tasks from another route, with $\ell_1 \leq \ell_2$. The local–search procedure starts with $\ell_1 = \ell_2 = 1$, and tries to interchange the tasks between the two routes in order to find an improving move. If no exchange move reduces the total time satisfying the route capacity and time limit constraints, ℓ_2 is incremented 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. The procedure stops when an improving move is found, or if $\ell_1 = \ell_2 = \ell_{max}$ and there are no exchange moves that produce a better solution.

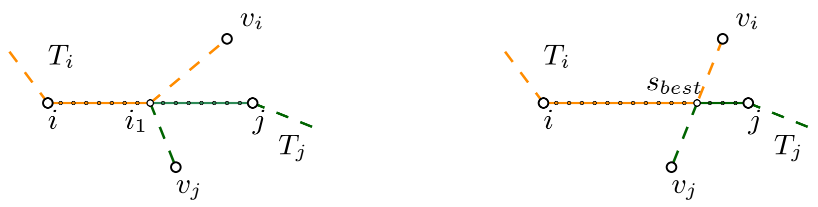
Once the VND algorithm is terminated, a *route–optimization* procedure is applied to each solution $\mathcal{T} \in \bar{\mathcal{T}}$. For each drone route T of a solution $\mathcal{T}$, this procedure defines a GRP instance on graph G formed by the required edges and the required nodes corresponding to the tasks performed on T. This GRP instance is then optimally solved with the branch–and–cut algorithm proposed in Corberán et al. (2007). The resulting routes will have a total time less than or equal to that of the original ones.

6.4.3 Adding intermediate points to obtain better K–GRP solutions

Let $\bar{\mathcal{T}}_b$ be the subset of the n best K–GRP(0) solutions obtained in the first part of the matheuristic. We add an intermediate vertex i_1 (equidistant from both endpoints) to each required edge (i, j) of G to obtain a new instance called K–GRP(1) with two edges $(i, i_1), (i_1, j)$ for each original required edge (i, j). Given a solution $\mathcal{T} \in \bar{\mathcal{T}}_b$ of K–GRP(0), it is easy to transform it into a solution $\mathcal{T}'$ of K–GRP(1) with the same total duration that traverses edges $(i, i_1), (i_1, j)$ consecutively.

Let $\bar{\mathcal{T}}_1$ be the set of all the K–GRP(1) solutions obtained in this way. The VND algorithm and the route optimization procedure are applied to each $\mathcal{T}' \in \bar{\mathcal{T}}_1$ to try to improve their overall duration. Observe that drones can now enter and leave any line through its middle point, which may lead to a better solution where the service of some original lines can be shared by two drones.

Let $E_1^{\mathcal{T}'}$ be the set of original lines whose middle point is incident to non–required edges in solution $\mathcal{T}'$. Note that an edge in $E_1^{\mathcal{T}'}$ is serviced by two drones (or by the same drone but not servicing both halves consecutively). It may be possible to improve this solution by "moving" this middle point closer to the extreme points of the edge. To do this, we consider p, with p odd, intermediate vertices evenly spaced in the line. Note that the middle point is one of them. Then, we study the improvement obtained by changing the entry/leaving point for this edge to each one of the $p - 1$ other new points. This procedure, which we call "p–splitting", is applied to all the edges in $E_1^{\mathcal{T}'}$.



(a) Adding p new intermediate points (b) Selecting the best intermediate point

Figure 6.4: Illustration of p–splitting

Figure 6.4a illustrates how the "p–splitting" procedure improves a solution where two (not necessarily different) routes, T_i (in orange) and T_j (in green), are involved in servicing an original required line joining vertices i and j. Note that these routes enter or leave the line at its middle point i_1 and, thus, i_1 is incident with two non–required

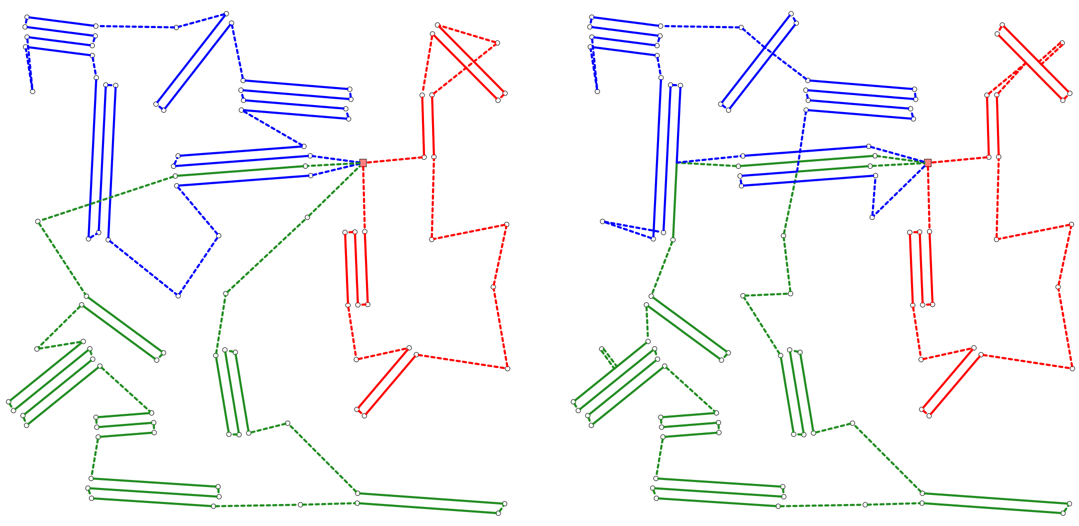


Figure 6.5: Two solutions of instance MPDGRP882 before and after applying the splitting part of the matheuristic

edges (dashed lines) in the solution. After analyzing the total duration of the routes obtained by replacing vertex i_1 by all the p intermediate points, the best solution is the one that uses vertex s_{best}, which is depicted in Figure 6.4b.

This p–splitting procedure is applied with $p = 15$ to each solution $\mathcal{T}' \in \bar{\mathcal{T}}_1$ and the best solution obtained is selected as the final solution of the matheuristic.

Figures 6.5a and 6.5b illustrate an example of a solution before and after applying the procedure described in this subsection. For this instance, the deadheading time is reduced by 5.2% due to drones entering and leaving six of the original required edges through some of their intermediate points.

6.5 Computational Experiments

In this section, we present the instances we have generated 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 2.8 GHz with 16 GB RAM. The B&C uses CPLEX 12.10 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 used for obtaining the initial optimal giant tour and for optimizing the routes after each VND improvement phase was also coded in C++ and uses CPLEX 12.10 MIP Solver too.

6.5.1 Instances

As defined above, a Multi–Purpose K–Drones GRP instance is given by several sets of parallel lines (each set covering a continuous area), some isolated vertices for delivery (those required nodes with positive demand), and a depot. In order to evaluate the behavior of the branch and cut and the matheuristic algorithm, we have randomly generated two different types of instances, both differing in the way they are built, classified as Type I or Type II instances.

First, we select values for n and m, and generate a GRP instance on a grid with $n \times m$ points whose coordinates are multiples of 100. The graph initially contains all the vertical and horizontal edges, as well as some random diagonals. For type I instances generation, we divide the grid into a number of regions computed by $nareas = \lceil \frac{n}{3} \rceil \cdot \lceil \frac{n}{3} \rceil$ and we randomly declare required an edge in each region. Each one of these required edges gives rise to a required area later. This procedure ensures that areas are disjoint and homogeneously distributed throughout the grid.

For type II instances generation, each edge of the original grid is considered required (and therefore included on the instance) with probability p, thus obtaining several connected components of required edges. As some of these components may contain more than one edge, we iteratively remove required edges that are incident to vertices with degree greater than one (at random) until we only have isolated required edges. The idea behind this second type is to generate larger instances with a set of required areas that are closer to each other. These type of instances can also be more difficult for our algorithms.

Once a representative required line from each area is generated, the following steps are identical for both types of instance. We randomly select the depot and a number *nvreq* of isolated required nodes among those vertices of the grid that are not incident with the selected required edges. The vertices, except for the depot, that are not required nor incident with required edges are removed, and the coordinates of all the vertices of the instance are randomly perturbed by adding a value in $[-20, 20]$ to each coordinate, avoiding completely horizontal and vertical edges and also slightly changing the length of the edges. Each required area is completely defined by adding a set of $npar(e)$ parallel lines to each initial required edge e (separated by a distance *distpar* to each other). The length and position of these new edges are then slightly perturbed in a random way so that the represented area has a more irregular shape. The demand of each required node v is given by a value $dem(v)$ and the service time of each required edge is the Euclidean distance multiplied by a parameter *timefactor*.

We have generated MP K–DGRP instances with different values for $n, m \in \{6, 7, 8, 9, 10, 12\}$, $npar(e) \in \{1, 2, 3\}$ for each initial required edge e, $distpar = 20$ and $timefactor = 1.5$. For type I instances, we have generated two sets with $nvreq \in [n-4, n+4]$: one with $dem(v) \in \{1, 2, 3\}$ for each required node v (version 1), and another with unit demands (version 2). Since the demands of the vertices represent the weight of the delivery, and given that it seems reasonable that the total weight that can be carried by a drone is not very large, we have not considered demands greater than 3. For type II instances, we have also generated two different sets with unit demands: one with $p = 0.4$ and $nvreq \in [n-1, n+1]$ (version 1), and another with $p = 0.3$ and $nvreq \in [2n-1, 2n+1]$ (version 2).

Instance name	$\|V\|$	$\|V_R\|$	D	$\|E_R\|$	♯ areas
MPDGRP661	31	6	14	12	4
MPDGRP662	37	6	6	15	4
MPDGRP681	48	7	16	20	6
MPDGRP682	50	7	7	21	6
MPDGRP771	32	5	10	13	4
MPDGRP772	34	5	5	14	4
MPDGRP861	50	7	15	21	6
MPDGRP862	55	8	8	23	6
MPDGRP881	32	5	10	13	4
MPDGRP882	39	6	6	16	4
MPDGRP8101	79	12	24	33	12
MPDGRP8102	93	10	10	41	12
MPDGRP991	61	10	22	25	9
MPDGRP992	69	10	10	29	9
MPDGRP1081	83	12	24	35	12
MPDGRP1082	85	12	12	36	12
MPDGRP10101	61	10	22	25	9
MPDGRP10102	66	11	11	27	9
MPDGRP12121	103	10	22	46	16
MPDGRP12122	111	12	12	49	16

(a) Type I instances

Instance name	$\|V\|$	$\|V_R\|$	$\|E_R\|$	♯ areas
MPDGRP661	60	7	26	8
MPDGRP662	54	13	20	6
MPDGRP681	83	6	38	11
MPDGRP682	70	13	28	9
MPDGRP6101	114	7	53	17
MPDGRP6102	85	12	36	14
MPDGRP771	105	8	48	14
MPDGRP772	71	14	28	9
MPDGRP791	134	7	63	20
MPDGRP792	81	14	33	11
MPDGRP881	118	9	54	18
MPDGRP882	103	16	43	15
MPDGRP8101	150	9	70	24
MPDGRP8102	97	16	40	12
MPDGRP991	157	10	73	25
MPDGRP992	74	19	27	11
MPDGRP9101	163	10	76	23
MPDGRP9102	94	19	37	14
MPDGRP10101	179	10	84	28
MPDGRP10102	143	20	61	19

(b) Type II instances

Table 6.1: Characteristics of the MP *K*–DGRP instances

The characteristics of all the MP *K*–DGRP instances generated are shown in Table 6.1 and can be found in http://www.uv.es/plani/instancias.htm. Table 6.1a shows, for each MP *K*–GRP instance of type I, the number of vertices, the number of required vertices and its total demand, the number of required lines, and the number of areas. Table 6.1b shows the type II instances characteristics and presents the same structure except for the total demand, which is not included because its value matches the number of required vertices. The digits in the name of each instance indicate the values of m, n and if this is the first or the second version generated from the same grid. Examples of a type I instance, a type II instance (version 1), and a type II instance (version 2) on a grid with $n = m = 9$ are shown in Figures 6.6a, 6.6b, and 6.6c, respectively.

In order to choose the values for L, each instance has been executed several times with different L values to guarantee that the solutions will use a number of drones ranging from 2 to 6. The capacity Q of the drones has been chosen so that all the demand of the required vertices can be serviced while keeping the number of packages carried by each drone as low as possible.

6.5.2 Computational results

We present here the results obtained with the branch and cut and the matheuristic algorithms on the MP *K*–DGRP instances. Note that the branch–and–cut (B&C) algorithm is applied only on the MP *K*–DGRP instances without adding any additional intermediate vertices, i.e. the so–called *K*–GRP(0) instances.

Each of these 40 instances, 20 of type I and 20 of type II, is solved with 2, 3, 4, 5 and 6 drones, and, hence, a total of 200 instances have been run.

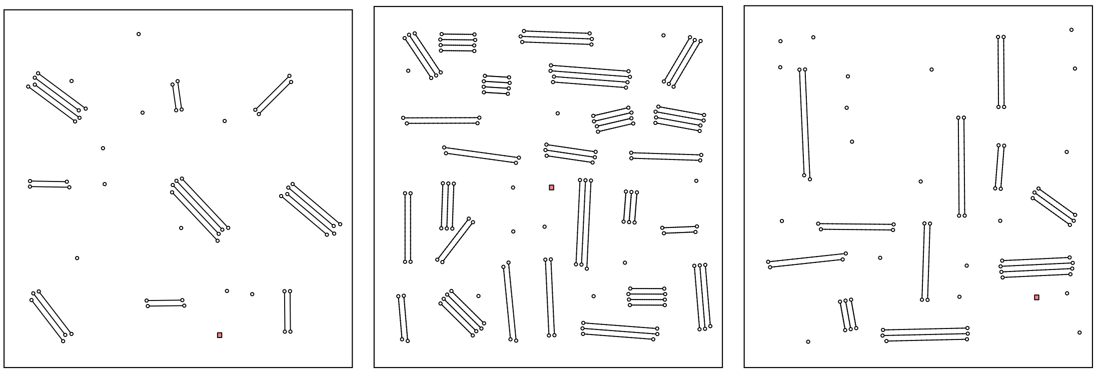

(a) Type I (b) Type II (Version 1) (c) Type II (Version 2)

Figure 6.6: MP K–DGRP instances on a grid 9×9

Tables 6.2 and 6.3 summarize the computational results obtained with both algorithms for all the instances of type I and type II, respectively. Both tables present the same structure. The results for each type of instance are separated in two blocks by a double horizontal line according to the instance generation characteristics (general or unit demands in type I instances, and version 1 or version 2 in type II instances). Each of these blocks is also separated according to the number of vertices. The number of drones used by the solution is shown in Column 2. Each row refers to the data of 5 instances.

The results obtained with the B&C algorithm for the corresponding K–GRP(0) instances with a time limit of 7200 seconds are reported in columns 3 to 7. Column 3 shows the number of instances out of five solved to optimality. Columns 4 and 5 show the average percentage gaps between the value of the optimal solution (or the best upper bound found) and the lower bound at the end of the root node ("Gap0") and the final lower bound ("Gap"), respectively. Column 6 reports the average number of nodes of the branching tree and Column 7 shows the average total time, in seconds, used by the B&C.

From Table 6.2 we observe that the B&C is capable of solving all type I instances with up to 60 vertices and 2, 3, 4, 5, and 6 drones in less than 30 minutes on average of computing time. Regarding the instances with more than 60 vertices, it can be seen that most of the instances with 2, 3, and 4 drones (26 out of 30) have been solved in less than one hour of computing time. However, the B&C has only been able to solve 3 out of 20 instances with 5 and 6 drones. On the other hand, comparing the values of "Gap0", "Gap", and "Time" obtained for the instances with general demands and unitary demands, we can observe that the latter seem a bit more difficult for the B&C than those with general demands. This apparently small increase in difficulty could be explained by the use of CPLEX cover inequalities in the case of general demands.

Table 6.3 reports the results obtained by the B&C and the matheuristic in Type II instances, which are associated with larger graphs. It can be seen that only a small number of instances have been solved to optimality. Specifically, 17 instances out of 20 with 2 drones, 12 out of 20 with 3 drones, and 6 with 4, 5, and 6 drones out of 60 instances. Note, however, that except for the largest instances with 6 drones, the final gaps obtained are very good, showing that the inequalities in the formulation and

			Branch–and–Cut					**Matheuristic**				
	$\|V\|$	K	♯ opt	Gap0 (%)	Gap (%)	Nodes	Time	♯ opt	♯ best	Gap (%)	Imp (%)	Time
General Demands		2	5	3.86	0.00	57.0	3.9	5/5	5	0.00	0.19	3.4
		3	5	5.82	0.00	178.2	16.3	4/5	4	0.13	0.02	3.9
	≤ 60	4	5	8.78	0.00	237.2	24.8	5/5	5	0.00	0.11	2.9
		5	5	12.72	0.00	654.6	98.5	4/5	4	0.00	0.67	3.7
		6	5	17.02	0.00	2590.8	427.0	3/4	3	0.13	0.65	4.4
		2	5	3.00	0.00	199.0	49.1	4/5	4	0.00	0.15	30.1
		3	5	5.22	0.00	705.6	349.9	2/5	2	0.09	0.02	19.9
	> 60	4	3	8.80	0.82	5498.6	3828.4	2/3	4	0.98	0.26	19.9
		5	2	10.07	2.46	4224.0	5501.0	1/2	3	2.67	0.57	17.8
		6	0	11.24	4.50	5042.6	7200.0	0/0	3	4.97	0.20	17.4
Unit Demands		2	5	3.17	0.00	46.2	6.3	4/5	4	0.01	0.01	4.8
		3	5	6.47	0.00	206.4	25.5	5/5	5	0.00	0.01	4.1
	≤ 60	4	5	10.75	0.00	348.8	60.2	3/5	3	1.28	0.10	4.2
		5	5	15.17	0.00	1991.4	1511.9	5/5	5	0.00	0.26	4.5
		6	5	17.74	0.00	4065.2	1736.5	4/5	4	0.02	0.83	4.4
		2	5	3.95	0.00	608.8	112.7	3/5	3	0.28	0.11	43.7
		3	5	5.11	0.00	381.0	148.7	3/5	3	0.21	0.02	34.4
	> 60	4	3	8.34	1.02	2899.4	3211.5	1/3	3	1.04	0.08	28.0
		5	1	12.01	2.73	4659.8	5692.9	1/1	4	2.83	0.28	26.9
		6	0	13.18	5.03	4196.2	7200.0	0/0	5	5.03	0.24	27.5
Total			**79**					**59/79**	**76**			

Table 6.2: Computational results with the B&C and the matheuristic on the instances of Type I

the proposed valid inequalities provide a good description of the convex hull of the solutions of the problem. These results reflect the great difficulty of the problem when the instance is large and encourage the development of heuristic algorithms for solving the multi–purpose K–drones general routing problem.

The results obtained with the matheuristic on all the instances of type I and II are reported in columns 8 to 12 of Tables 6.2 and 6.3. To analyze its performance, we have compared the results of the first part of the matheuristic (with no splitting) with those of the branch and cut on the K–GRP(0) instances. Note that the final solutions of the matheuristic (provided after the second part of the algorithm) are not feasible solutions of the K–GRP(0) instance, but we compare them with the solutions obtained in the first part to be able to analyze the improvement of allowing drones to use some intermediate vertices.

Column 8 shows the number of optimal solutions of the K–GRP(0) instances found with the matheuristic, and column 9 reports the number of times the matheuristic reaches the optimal solution or provides the best upper bound. The "Gap" column shows the average percentage gap between the value of the solution provided by the first part of the matheuristic and the lower bound given by the branch–and–cut algorithm in two hours of computing time. Column "Imp" gives the average percentage improvement after applying the second part of the matheuristic algorithm with respect to the solution provided in the first part. The last column reports the average total computing time, in seconds, used by the matheuristic.

We can see that in 151 out of the 200 instances considered, the matheuristic provides

			Branch–and–Cut					**Matheuristic**				
	$\|V\|$	K	♯ opt	Gap0 (%)	Gap (%)	Nodes	Time	♯ opt	♯ best	Gap (%)	Imp (%)	Time
Version 1	≤ 120	2	5	2.69	0.00	2775.2	761.6	5/5	5	0.00	0.30	68.2
		3	4	4.08	0.07	5702.8	4240.7	1/4	2	0.15	0.38	58.0
		4	2	5.00	1.48	4610.2	5792.5	2/2	5	1.48	0.48	46.0
		5	0	7.35	4.56	3594.0	7200.0	0/0	5	4.56	0.69	44.3
		6	0	8.58	6.02	2892.4	7200.0	0/0	5	6.02	0.66	39.7
	> 120	2	2	2.01	0.33	3746.2	4750.6	1/2	2	0.51	0.26	285.0
		3	0	3.03	1.66	2535.4	7200.0	0/0	2	1.83	0.41	215.4
		4	0	4.02	2.80	1303.2	7200.0	0/0	5	2.80	0.18	170.0
		5	0	5.45	4.51	814.2	7200.0	0/0	4	4.51	0.25	162.0
		6	0	7.46	6.77	499.0	7200.0	0/0	5	6.77	0.45	154.3
Version 2	≤ 84	2	5	2.86	0.00	491.2	44.8	2/5	2	0.04	0.38	24.5
		3	5	4.71	0.00	2947.2	1521.3	2/5	2	0.23	0.39	19.3
		4	3	5.99	0.90	4334.0	4362.8	1/3	3	1.19	0.84	19.2
		5	1	9.21	3.70	3694.6	7200.0	1/1	5	3.70	0.84	18.6
		6	0	10.67	6.07	2545.0	7200.0	0/0	5	6.07	0.71	19.0
	> 84	2	5	2.29	0.00	1144.6	283.9	1/5	1	0.40	0.44	130.9
		3	3	4.13	0.44	4166.2	4446.0	1/3	3	0.57	0.43	82.6
		4	0	6.28	3.47	2591.8	7200.0	0/0	4	3.55	0.40	89.3
		5	0	9.41	6.99	1504.0	7200.0	0/0	5	6.99	0.52	71.9
		6	0	11.04	8.94	1017.2	7200.0	0/0	5	8.94	0.20	70.3
Total			**35**					**17/35**	**75**			

Table 6.3: Computational results with the B&C and the matheuristic on the instances of Type II

a K–GRP(0) solution equal to or better than the one obtained with the B&C in a much more reasonable computing time. Nearly 67 % of the K–GRP(0) instances optimally solved by the B&C are also optimally solved by the matheuristic.

Table 6.4 summarizes the results obtained by the matheuristic on the 114 instances for which an optimal solution is known for the corresponding K–GRP(0) instance. Columns 1 and 2 contain the instance type and the number of drones. Column 3 reports the number of K–GRP(0) instances with known optimal value, and Column 4 the number among them for which the first part of the matheuristic provides the optimal solution. The “Gap” column shows the average percentage gap between the value of the solution of the first part of the matheuristic and the optimal solution, while “Gap–p” represents the same gap for the solution obtained by the matheuristic after the p–splitting phase. Note that some of these latter gaps may be negative, since the solutions provided by the matheuristic for the instance with intermediate vertices may be better than the optimal solution of the K–GRP(0) instance. The “Time” column reports the average computing time in seconds.

As mentioned above, the first part of the matheuristic is able to optimally solve 76 out of the 114 instances for which the optimal solution is known. For the remaining 38, the solutions obtained are very close to the optimal ones. In addition, the p–splitting procedure allows us to improve them even more, obtaining solutions in many cases better than the optimal solutions without splits in very short computing times. This confirms that considering intermediate vertices when solving the problem can lead to better solutions.

Finally, to get an idea of how much the cost of the solutions is improved by using

Type	K	♯ opt B&C	♯ opt M	Gap (%)	Gap–p (%)	Time
I	2	20	16	0.07	−0.04	20.5
II		17	9	0.15	−0.23	95.9
I	3	20	14	0.11	0.09	15.6
II		12	4	0.18	−0.27	51.1
I	4	16	11	0.47	0.33	9.8
II		5	3	0.29	−0.55	24.4
I	5	13	11	0.00	−0.37	6.0
II		1	1	0.00	−0.56	9.0
I	6	10	7	0.08	−0.69	4.4
II		0	–	–	–	–
Total		**114**	**76**			

Table 6.4: Results for the instances with known optimal solutions

			Single–purpose			**Multi–purpose**	
	K	♯ inst	nodes	edges	nodes+edges	total cost	Imp (%)
General	2	5	3331.2	4377.4	7708.6	6146.1	20.35
	3	5	3916.4	4608.9	8525.3	6764.3	20.79
	4	5	4372.1	4901.2	9273.3	7362.2	20.70
	5	5	4711.8	5424.9	10136.7	8269.4	18.28
	6	5	4907.5	6092.0	10999.5	9203.1	15.71
Unit	2	5	3066.0	4359.3	7425.3	5830.9	21.63
	3	5	3239.5	4667.9	7907.4	6306.9	20.24
	4	5	3819.6	5022.1	8841.7	7009.1	20.47
	5	5	3819.6	5512.5	9332.0	7954.6	14.41
	6	5	4225.1	6130.6	10355.7	8777.8	14.87

Table 6.5: Comparison of results with single–purpose versus multi–purpose drones

multi–purpose drones versus using single–purpose drones (to service nodes or edges), we have applied our B&C on all the K–GRP(0) instances of Type I with $|V| \leq 60$, first considering only the required vertices of the instances and then only the required edges. We assume that two fleets of drones (K drones with delivery capability and K with inspection capability) are available to perform each type of task (it may be the same K drones that are reconfigured for missions of different type). The results obtained in these 50 instances are summarized in Table 6.5. Each row of this table reports the average data for the 5 solved instances. Columns 3 and 4 provide the total cost (on average) of the solutions if we consider drones that only service the required nodes (delivery only) and drones that only traverse all the required edges (sensing only), respectively. The sum of both of these costs for single purpose drones is shown in Column 5. The average costs of the solutions obtained with multi–purpose are shown in Column 6, and the last column gives the average percentage improvement that these costs represent with respect to the sum of the costs provided by single–purpose drones. It can be seen that the use of a fleet of multi–purpose drones improves the total cost between 14% and 21% (on average) compared to the use of two fleets of single–purpose drones.

Chapter 7

The load–dependent drone general routing problem

In this chapter, we address a new variant of the general routing problem (GRP). This combinatorial optimization problem, introduced in Orloff (1974), aims to find the minimum cost tour on a graph $G = (V, E)$ traversing each required edge $e \in E_R$ and visiting each required node $v \in V_R$ at least once. The extension we study here considers a (multi–purpose) drone that needs to be routed to both provide sensing over a set of edges and deliver goods to a set of vertices of the network. In the GRP, it is usually assumed that the cost of traveling from a node i to a node j in the graph is a constant c_{ij}. Nevertheless, the real cost of a vehicle which travels between any pair of nodes can depend on many other aspects, such as the load carried by the vehicle and the fuel or energy consumption.

In transportation systems with drones, the weight of the transported cargo represents a significant part of the gross weight of the vehicle and can decisively influence the battery consumption and the flight range of the drone (see Zhang et al., 2021), as well as the travel times of the edges of the network and the takeoff and landing times. It is therefore important to consider the weight carried by the drone at each moment of the route. Such a variation of the load weight along the trip is taken into account in our problem to determine the minimum duration drone route. Considering load dependency adds considerable difficulty to modeling and solving routing problems, since costs and travel times on the network are no longer constant.

Some routing problems that consider load–dependent costs have been studied in the literature. Kara et al. (2007) introduce the energy minimizing vehicle routing problem, a variant of the capacitated VRP that considers a new cost function based on distance and load of the vehicle, and propose integer programming formulations for both collection and delivery cases. The pollution routing problem (PRP) is introduced in Bektaş and Laporte (2011) as an extension of the VRP aimed at reducing the greenhouse gas emissions, which depend on the load of the truck. An exact approach for a variant of the PRP is proposed in Dabia et al. (2017). Another variant with load–dependent costs in the objective function is introduced in Zachariadis et al. (2015), where the VRP with pick–ups and deliveries is generalized. Regarding arc routing problems, Corberán et al. (2018) introduce load–dependent costs for the Chinese postman problem, providing two mathematical programming formulations and two metaheuristic algorithms to solve this difficult problem.

Load–dependent costs and travel times have also been considered in vehicle routing problems with drones and other green vehicles, such as cargo bicycles (Fontaine, 2022). A recent survey on last–mile drone delivery research can be found in Eskandaripour and Boldsaikhan (2023). Torabbeigi et al. (2020) address the design of a parcel delivery system using drones, showing that there is a linear relationship between the payload amount of the drone and the battery consumption rate. Dukkanci et al. (2021) introduce the energy–minimizing and range–constrained drone delivery problem, in which the speed of the drone is considered as a decision variable, motivated by its direct impact on the range and energy consumption of the drone.

We introduce and study in this chapter the Load–Dependent drone General Routing Problem (LDdGRP). The goal of this problem is to find a minimum duration tour for a drone with sensing and delivering capability that, starting and ending at a given depot, traverses a set of required edges of a network and also delivers goods at a set of required nodes. In the LDdGRP, it is assumed that the traversal time of each edge, as well as the takeoff and landing times, are proportional to the product of the distance travelled and the total weight (including cargo) of the drone.

As in other arc routing problems studied in previous chapters, drones can travel in a straight line between any two points of the network, and the optimization problem is continuous and very difficult to solve. These problems can be discretized by approximating each original line of the network by one or several required edges and allowing drones to enter and exit these edges only at their endpoints. The original characteristics of drone arc routing problems that remain in the discrete problem are that the set of non–required edges induce a complete graph (thus, each required edge has a parallel non–required one) and that the deadheading times of the network satisfy the triangular inequality. Unlike other chapters, we will focus here on formulating the discrete problem and proposing an exact algorithm for its solution, without addressing the issue of including and eliminating intermediate points on the original lines to improve the solution of the continuous problem.

The content of this chapter is organized as follows. A formal description for the LDdGRP is presented in Section 7.1. We propose in Section 7.2 a mathematical formulation for the LDdGRP. In Section 7.3, its associated polyhedron is studied, and several families of valid inequalities are proposed. In Section 7.4, we present a branch–and–cut algorithm for the LDdGRP that includes new separation procedures for the identification of violated inequalities. The set of generated instances and the computational experiments carried out to show the performance of the proposed algorithm are provided in Section 7.6.

7.1 Problem definition and notation

The LDdGRP is defined on an undirected multigraph $G = (V, E)$ with node set V and edge set $E = E_R \cup E_{NR}$. The set of required edges $E_R \subset E$ contains all the edges of the network that must be traversed (and serviced) by the drone at least once, and the set of non–required edges $E_{NR} \subseteq E$ forms a complete graph with the node set V. For each

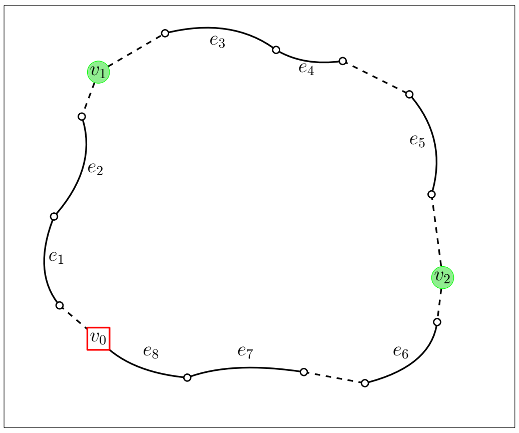
E	set of edges of the undirected network
E_R	set of required edges
E_{NR}	set of non–required edges
V	set of nodes of the undirected network
v_0	depot node
V_R	set of required nodes
V_R^0	$V_R \cup \{v_0\}$
V_I	set of nodes incident with required edges
d_i	non–negative demand of node i, for $i \in V_R$
$\mathcal{Q}$	total demand
w_0	curb weight of the drone
c_e^s	time (per unit of weight) of traversing and servicing edge $e \in E_R$
c_e	time (per unit of weight) of traversing/deadheading edge $e \in E$
c_D	time (per unit of weight) of landing and taking off when a vertex is serviced

Table 7.1: Notation used in this chapter

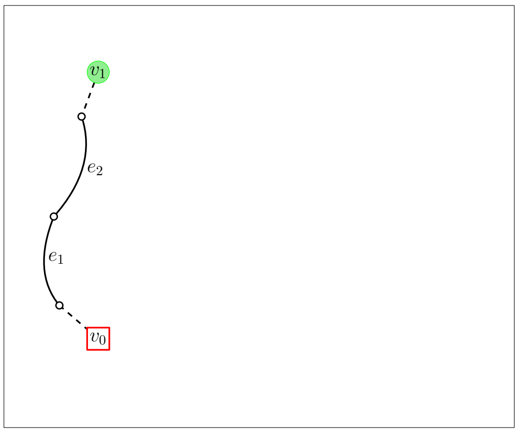
required edge $e \in E_R$, there is a non–required parallel edge in E_{NR} denoted by e'. The vertex set V contains the depot v_0, where the drone is initially located, a set V_I formed by those vertices that are incident with required edges, and a set of required vertices $V_R \subset V$, which is formed by all the nodes of V that must be visited (and serviced exactly once) by the drone in order to make a delivery. We assume without loss of generality that the required vertices and also the depot are not incident with required edges, and that $V = \{v_0\} \cup V_I \cup V_R$. For notation convenience, we will denote $V_R^0 = V_R \cup \{v_0\}$.

For each node $i \in V_R$, we are given a non–negative value d_i that specifies the demand at node i. We assume here that the drone has no load nor autonomy (flight range) limitation, so that a single vehicle can perform the entire service. The total demand is denoted by $\mathcal{Q} = \sum_{i \in V_R} d_i$. Each required edge $e \in E_R$ has an associated value $c_e^s \geq 0$ corresponding to the time of traversing and servicing it for each unit of total weight. Therefore, if the drone has a curb weight w_0 and carries a load weighting q at the moment of servicing edge e, the duration of traversing and servicing it is given by $(w_0 + q) \cdot c_e^s$. Similarly, each non–required edge $e \in E_{NR}$ has an associated deadheading time c_e for each unit of total weight. These deadheading times satisfy the triangular inequality, and we assume $c_e^s \geq c_{e'}$ for each required edge e and its corresponding parallel edge $e' \in E_{NR}$. Moreover, the time taken by the drone to land and take off at a vertex in V_R (and to land or take off at v_0) also depends on the total weight of the drone. This can also be modeled as $(w_0 + q) \cdot c_D$, where c_D is a parameter representing the time (per unit of weight) of the drone taking off and landing. We summarize the notation employed in Table 7.1.

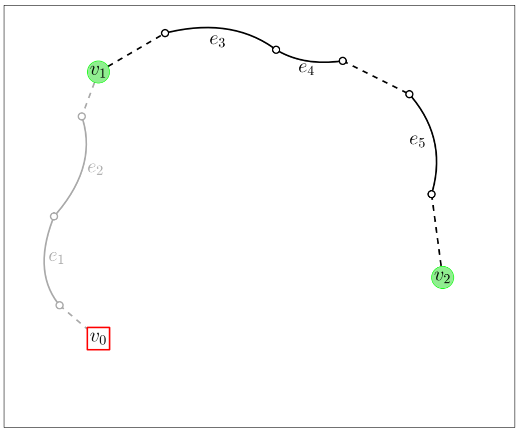
The objective of the LDdGRP is to find a route with minimum total duration, starting and ending at the depot, traversing all the required edges and visiting each required node exactly once.



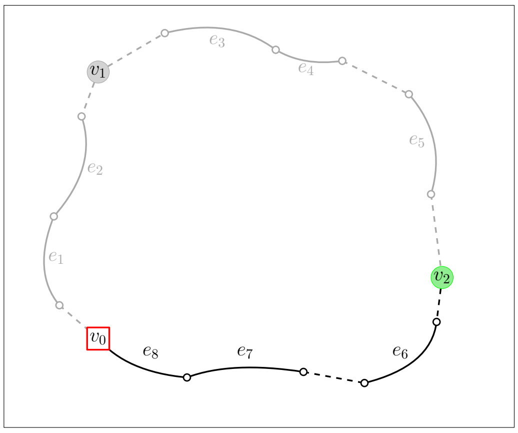
(a) An LDdGRP tour on G



(b) Path from v_0 to v_1



(c) Path from v_1 to v_2



(d) Path from v_2 to v_0

Figure 7.1: Example of a drone tour decomposition on $|V_R| + 1 = 3$ paths

7.2 A formulation for the LDdGRP

As defined above, an LDdGRP solution is a tour starting and ending at the depot, traversing (and servicing) all the required edges and visiting all the required nodes exactly once. Unlike classical general routing problems, the total duration of this LDdGRP tour depends on the weight with which the drone traverses each required or non–required edge of the tour and visits each required vertex. We propose in this section a formulation for the LDdGRP based on considering each solution as an ordered set of paths in G.

To first illustrate the idea behind such a formulation, consider an LDdGRP instance defined on a graph G with $|V_R| = 2$ and $|E_R| = 8$. The tour $T = (v_0, e_1, e_2, v_1, e_3, e_4, e_5, v_2, e_6, e_7, e_8, v_0)$ on G (see Figure 7.1a) is a feasible LDdGRP solution. In this tour, the drone starts at the depot (represented in Figure 7.1 by a red square node v_0), services edges e_1 and e_2, and then visits node v_1 to make the first delivery. In this part of the tour, a path from v_0 to v_1, the drone travels with full load $\mathcal{Q}$ (Figure 7.1b). Next, the tour contains a path from v_1 to v_2 with load $\mathcal{Q} - d_{v_1}$ that services edges e_3, e_4 and e_5 and visits node v_2 (Figure 7.1c). Once the drone is empty, it returns to the depot after servicing edges e_6, e_7 and e_8 by following a path from v_2 to v_0 (Figure 7.1d).

Hence, any feasible LDdGRP solution will be considered as an ordered set of $P = |V_R| + 1$ paths. The end of each of the first $P - 1$ paths is determined by the service of a (different) required vertex, while the last path of such set ends at the depot. Inside each one of these P paths, the load carried by the drone does not change (it remains constant from leaving a required node until visiting the next required node in the tour). Thus, the load in the first path is equal to the total demand Q, and in the last path the drone flies empty (all the deliveries have already been carried out). In what follows, we will call LDdGRP *solution* to any ordered set of P paths on graph G such that:

i) the first path starts at the depot,

ii) each path $p \in \{2, \dots, P\}$ starts at the final vertex of path $p - 1$,

iii) each path $p \in \{1, 2, \dots, P - 1\}$ ends at a different node of V_R,

iv) the last path P ends at the depot, and

v) they jointly traverse all the required edges of the network.

Note that such a definition guarantees that each required vertex is visited by the tour. In order to formulate the problem, we define the following decision variables. For each path $p \in \{1, \dots, P\}$ and each node $i \in V_R^0$, we define a binary variable z_i^p that takes value 1 if node i is the end node of path p (and, hence, the initial node of path $p+1$). For each path $p \in \{1, \dots, P\}$ and each required edge $e \in E_R$, we define a binary variable x_e^p that takes value 1 if e is traversed in the p–th path, and 0 otherwise. We also define for each path $p \in \{1, \dots, P\}$ and each non–required edge $e \in E_{NR}$ two binary variables x_e^p and y_e^p representing the first and the second traversal of e in path p. Thus, x_e^p takes value 1 if $e \in E_{NR}$ is deadheaded at least once in path p and y_e^p takes value 1 if it is deadheaded twice in the p–th path. Furthermore, to represent the weight of the load carried by the drone on each path $p \in \{1, \dots, P\}$, we introduce a variable q^p. Table 7.2 summarizes the above defined variables. We denote by $\mathcal{P}$ the set of indices $\{1, ..., P\}$ referring to the paths.

z_i^p	1, if node $i \in V_R^0$ is the end–node of path $p \in \mathcal{P}$, and 0, otherwise
x_e^p	1, if edge $e \in E_R$ is traversed in path $p \in \mathcal{P}$, and 0, otherwise
x_e^p	1, if edge $e \in E_{NR}$ is traversed in path $p \in \mathcal{P}$, and 0, otherwise
y_e^p	1, if edge $e \in E_{NR}$ is traversed twice in path $p \in \mathcal{P}$, and 0, otherwise
q^p	weight carried by the drone in path $p \in \mathcal{P}$

Table 7.2: Decision variables in the LDdGRP formulation

For notational convenience we consider, for each $i \in V_R^0$, an additional parameter z_i^0 that is fixed to zero except for $z_{v_0}^0 = 1$ to indicate that the first path (and therefore the tour) must start at the depot. We will use the following notation. Given a subset of vertices $S \subseteq V$, $\delta(S) = (S : V \setminus S)$ denotes the edge set with one endpoint in S and the other in $V \setminus S$, 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 of edges $F \subseteq E$, we denote $F_R = F \cap E_R$, $F_{NR} = F \cap E_{NR}$, $x^p(F) = \sum_{e \in F} x_e^p$ and $(x^p + y^p)(F) = \sum_{e \in F}(x_e^p + y_e^p)$, and write $\delta_R(S) = \delta(S) \cap E_R$, instead of $\delta(S)_R$, and $\delta_{NR}(S) = \delta(S) \cap E_{NR}$, for any $S \subseteq V$.

The load–dependent drone general routing problem can be formulated as follows:

$$\text{Minimize} \quad \sum_{p\in\mathcal{P}}\sum_{e\in E_{NR}} c_e\,(w_0+q^p)\,(x_e^p+y_e^p)+\sum_{p\in\mathcal{P}}\sum_{e\in E_R} c_e^s\,(w_0+q^p)\,x_e^p+\sum_{p\in\mathcal{P}} c_D\,(w_0+q^p) \tag{7.1}$$

s.t.

$$x^p(\delta_R(i))+(x^p+y^p)(\delta_{NR}(i))\equiv 0 \pmod 2, \qquad \forall i\in V_I,\ \ \forall p\in\mathcal{P}, \tag{7.2}$$

$$(x^p+y^p)(\delta_{NR}(i))+z_i^{p-1}+z_i^p\equiv 0 \pmod 2, \qquad \forall i\in V_R^0,\ \ \forall p\in\mathcal{P}, \tag{7.3}$$

$$x^p(\delta_R(S))+(x^p+y^p)(\delta_{NR}(S))\geq 2x_f^p-\sum_{i\in S\cap V_R^0}\left(z_i^{p-1}+z_i^p\right), \qquad \forall S\subset V,\ \forall f\in E(S),\ \forall p\in\mathcal{P}, \tag{7.4}$$

$$\sum_{p\in\mathcal{P}} x_e^p\geq 1, \qquad \forall e\in E_R, \tag{7.5}$$

$$\sum_{p\in\mathcal{P}} z_i^p=1, \qquad \forall i\in V_R^0, \tag{7.6}$$

$$\sum_{i\in V_R^0} z_i^p=1, \qquad \forall p\in\mathcal{P}, \tag{7.7}$$

$$z_{v_0}^P=1, \tag{7.8}$$

$$q^1=\mathcal{Q}, \tag{7.9}$$

$$q^p=q^{p-1}-\sum_{i\in V_R} d_i z_i^{p-1}, \qquad \forall p\geq 2, \tag{7.10}$$

$$x_e^p\geq y_e^p, \qquad \forall e\in E_{NR},\ \ \forall p\in\mathcal{P}, \tag{7.11}$$

$$z_i^p,\in\{0,1\}, \qquad \forall i\in V_R^0,\ \forall p\in\mathcal{P}, \tag{7.12}$$

$$x_e^p\in\{0,1\}, \qquad \forall e\in E_R,\ \ \forall p\in\mathcal{P}, \tag{7.13}$$

$$x_e^p,y_e^p\in\{0,1\}, \qquad \forall e\in E_{NR},\ \ \forall p\in\mathcal{P}. \tag{7.14}$$

The objective function (7.1) minimizes the total duration of the paths. The first term represents the deadheading time of traveling on non–required edges. The second and third terms represent, respectively, the time of servicing the required edges and the time of visiting the required nodes (for delivery).

Constraints (7.2) force that the number of times each path visits a vertex incident with a required edge is even, possibly zero. These constraints are not linear, but we can use instead the following *parity inequalities*, which are obtained from those proposed in Corberán et al. (2013):

$$x^p(\delta_R(S)\backslash F_R)+(x^p-y^p)(\delta_{NR}(S)\backslash F_{NR})\geq x^p(F_R)+(x^p-y^p)(F_{NR})-|F|+1, \tag{7.15}$$

for each path $p\in\mathcal{P}$, for each $S\in V_I$, and for all $F\subseteq\delta(S)$ with $|F|$ odd. These inequalities ensure the parity (even degree) of every subset of vertices and, in particular, at every vertex in V_I. Note that they impose that if a drone path traverses an odd number of edges incident to a set S of vertices in V_I, then the path uses at least one additional traversal of some edge in the cut–set $\delta(S)$.

Constraints (7.3) force the number of edges on the global tour incident with the depot and with each required node to be even. Note that, for any optimal solution, a required vertex will only be visited at the beginning or end of a path. Therefore, these constraints can be replaced by

$$(x^p + y^p)(\delta_{NR}(i)) = z_i^{p-1} + z_i^p, \quad \forall i \in V_R^0, \ \ \forall p \in \mathcal{P}, \tag{7.16}$$

which are satisfied by all the optimal solutions.

Inequalities (7.4) guarantee that variables x^p and y^p define a connected path joining two required vertices (or the depot). Given a subset of vertices S, if there is an edge $f \in E(S)$ traversed by path p and the beginning and ending vertices of this path do not belong to S, the right–hand side of the constraint takes value 2, and the inequality says that the cutset defined by S has to be traversed at least twice. If only the initial vertex of the path (or the final one) belongs to S, the right–hand side takes value 1, and the cutset has to be traversed at least once. Finally, if both extremes of the path are in S or there is no edge $f \in E(S)$ traversed by the path, the inequality is trivially satisfied since there is no obligation for path p to leave set S.

The following inequalities are obtained by replacing in (7.4) the edge $f \in E(S)$ by the z_i^p variable corresponding to a required vertex (or the depot) in S. They are also valid and can reinforce the formulation, although they are not necessary for it.

$$x^p(\delta_R(S)) + (x^p + y^p)(\delta_{NR}(S)) \geq 2z_i^p - \sum_{j \in S \cap V_R^0} \left(z_j^{p-1} + z_j^p \right), \forall S \subset V, \ \forall i \in V_R^0 \cap S, \ \forall p \in \mathcal{P} \tag{7.17}$$

Inequalities (7.5) force each required edge to be serviced at least once. It can be seen that there is always an optimal solution satisfying this constraint with equality, since for any required edge $e \in E_R$ there is a parallel non–required edge $e' \in E_{NR}$ with $c_{e'} \leq c_e^s$. Constraints (7.6) ensure that each required node, as well as the depot, is the end node of exactly one path on the solution, while constraints (7.7) force each path to end in exactly one vertex in V_R^0. Equality (7.8) guarantees the return to the depot in the last path P, while equalities (7.9) and (7.10) determine the weight carried by the drone on each path. Constraints (7.11) ensure that a second traversal of a non–required edge in a path can only occur when this edge has been traversed previously in this path. Finally, constraints (7.12), (7.13) and (7.14) are the binary conditions for the variables.

The objective function is non–linear because of the terms $q^p(x_e^p + y_e^p)$. For each edge $e \in E$ and each path $p \in \mathcal{P}$, we can define the variable $q_e^p = q^p x_e^p$ and for each edge $e \in E_{NR}$ and each path $p \in \mathcal{P}$, variable $\bar{q}_e^p = q^p y_e^p$, so that by adding the constraints

$$q_e^p \geq q^p + (x_e^p - 1)\mathcal{Q}, \qquad \forall e \in E, \ \forall p \in \mathcal{P} \tag{7.18}$$

$$q_e^p \geq 0, \qquad \forall e \in E, \ \forall p \in \mathcal{P} \tag{7.19}$$

$$\bar{q}_e^p \geq q^p + (y_e^p - 1)\mathcal{Q}, \qquad \forall e \in E_{NR}, \ \forall p \in \mathcal{P} \tag{7.20}$$

$$\bar{q}_e^p \geq 0, \qquad \forall e \in E_{NR}, \ \forall p \in \mathcal{P} \tag{7.21}$$

to the formulation, the first two terms of the objective function can be linearized as

$$\sum_{p\in\mathcal{P}}\sum_{e\in E_{NR}} c_e\Big(w_0\big(x_e^p+y_e^p\big)+q_e^p+\bar{q}_e^p\Big)+\sum_{p\in\mathcal{P}}\sum_{e\in E_R} c_e^s\Big(w_0x_e^p+q_e^p\Big).$$

The description of these additional decision variables is given next.

q_e^p	weight carried by the drone when it traverses edge e in path $p\in\mathcal{P}$, for $e\in E$
$\bar{q}_e^p$	weight carried by the drone in the second traversal of edge e in path $p\in\mathcal{P}$, for $e\in E_{NR}$

Table 7.3: Additional variables in the LDdGRP formulation

Some variables of this formulation can be fixed to 0. Note that $z_{v_0}^p=0$, for all path $p\neq P$, and $z_i^P=0$, for all $i\in V_R$, hold due to equations (7.6), (7.7) and (7.8). The following result allows us to remove some variables from the formulation, taking into account the structure of the optimal LDdGRP solutions.

Theorem 7.2.1. *Any optimal* LDdGRP *solution satisfies*

i) $y_e^p=0$*, for each* $e\in\delta(v)$*, with* $v\in V_R^0$*, and for each path* $p\in\mathcal{P}$*, and*

ii) $x_e^p=0$*, for each (non–required)* $e\in\delta(v_0)$ *and for each path* $p\in\{2,\dots,P-1\}$.

Proof: Let us consider a vertex $v\in V_R^0$ and a path $p\in\mathcal{P}$. Each optimal solution satisfies (7.16), $(x^p+y^p)(\delta(v))=z_v^{p-1}+z_v^p$, and from (7.6), $z_v^{p-1}+z_v^p\leq 1$. Hence, $y_e^p=0$ for all edge $e\in\delta(v)$ and we obtain i). Moreover, if $v=v_0$ and $p\neq 1, p\neq P$, then $z_{v_0}^{p-1}=z_{v_0}^p=0$ and $(x^p+y^p)(\delta(v_0))=0$ holds. Therefore, $x_e^p=0$ for each edge $e\in\delta(v_0)$ and we obtain ii). □

7.3 Polyhedral study of the LDdGRP

In this section, we study the polyhedron associated with the LDdGRP solutions, which are considered as a set of $P=|V_R|+1$ paths as described above. Each LDdGRP solution is represented by an incidence vector

$$(\mathbf{x},\mathbf{y},\mathbf{z})\in\mathbb{Z}^{P(|E_R|+2|E_{NR}|+|V_R|+1)},$$

where variable x_e^p takes the value 1 if edge $e\in E$ is traversed once in path $p\in\{1,\dots,P\}$, variable y_e^p takes the value 1 if edge $e\in E_{NR}$ is traversed twice, and variable z_i^p takes value 1 if vertex $i\in V_R^0$ is the last node visited in path p. Note that knowing the load carried by the drone when traversing an edge is not necessary to determine the feasibility of the solution. Therefore, we do not need the q variables to define the polyhedron of solutions.

We keep here the assumption that the LDdGRP is defined on a graph G with a non–required edge e' parallel to each required edge e. These non–required edges parallel to required ones form the set $E'_{NR}\subset E_{NR}$, while $E''_{NR}=E_{NR}\backslash E'_{NR}$. However, in order to make a more general study, we do not assume that (V, E_{NR}) forms a complete graph, although we assume graph (V_R^0, E_{NR}) is complete.

Before studying the dimension of the polyhedron of solutions, we need some previous results about the vectors $\mathbf{z} \in \mathbb{Z}^{P(|V_R|+1)}$ and $(\mathbf{x}, \mathbf{y}) \in \mathbb{Z}^{P(|E_R|+2|E_{NR}|)}$ associated with the LDdGRP solutions.

Lemma 7.3.1. *All the vectors* $\mathbf{z} \in \mathbb{Z}^{P(|V_R|+1)}$ *associated with* LDdGRP *solutions satisfy* $4|V_R|$ *linear independent equations.*

Proof: The following $4|V_R| + 1$ equalities

$$z_{v_0}^P = 1,$$
$$z_i^P = 0, \qquad \forall i \in V_R$$
$$z_{v_0}^p = 0, \qquad \forall p \leq P-1$$
$$\sum_{p=1}^{P-1} z_i^p = 1, \qquad \forall i \in V_R \tag{7.22}$$
$$\sum_{i \in V_R} z_i^p = 1, \qquad \forall p \leq P-1 \tag{7.23}$$

are satisfied by all the vectors $\mathbf{z}$ associated with LDdGRP solutions. The first $1+2|V_R|$ are obviously linearly independent. It can be seen that $2|V_R|-1$ of the last $|V_R|+P-1 = 2|V_R|$ equations are linearly independent. □

Note that equations (7.22) and (7.23) correspond to the constraints of an *assignment problem* in which each vertex $i \in V_R$ has to be assigned to a path $p \in \{1, \ldots, P-1\}$. It is known that these equations totally define the polyhedron of the assignment problem (see Balinski and Russakoff, 1974), and therefore the dimension of such a polyhedron is $|V_R|^2 - (2|V_R| - 1)$. Hence, there exist $|V_R|^2 - 2|V_R| + 2 = (|V_R| - 1)^2 + 1$ affinely independent vectors corresponding to solutions of the assignment problem. These solutions can be completed with variables $z_{v_0}^P = 1$, $z_i^P = 0, \forall i \in V_R$ and $z_{v_0}^p = 0, \forall p \leq P-1$ to obtain $\mathbf{z}$ vectors associated with LDdGRP solutions, and then the following result holds:

Theorem 7.3.2. *There are* $(|V_R| - 1)^2 + 1$ *affinely independent vectors* $\mathbf{z} \in \mathbb{Z}^{P(|V_R|+1)}$ *associated with* LDdGRP *solutions.*

In Section 4.2.1, the 1–RPP was defined on an undirected and connected graph $G = (V, E_R \cup E_{NR})$, with $E_{NR} = E'_{NR} \cup E''_{NR}$, where E'_{NR} is the set of non–required edges parallel to an edge in E_R, and where the set V_R formed with the vertices incident with some edge in E_R plus the depot is not necessarily equal to V. The 1–RPP was formulated in Chapter 4 with a binary variable x_e for each edge $e \in E_R$ representing the service of required edge e, and two binary variables x_e and y_e for each edge $e \in E_{NR}$ representing the first and second traversal of non–required edge e, respectively. It was shown in Theorem 4.2.2 that the convex hull of all the 1–RPP tours, i.e., the vectors (x, y) satisfying

$$\sum_{e\in\delta_R(i)} x_e + \sum_{e\in\delta_{NR}(i)} \big(x_e + y_e\big) \equiv 0 \pmod 2, \qquad \forall i \in V \tag{7.24}$$

$$\sum_{e\in\delta_R(S)} x_e + \sum_{e\in\delta_{NR}(S)} \big(x_e + y_e\big) \geq 2x_f, \qquad \forall S \subseteq V\setminus\{1\}, \forall f \in E(S), \tag{7.25}$$

$$x_e = 1, \qquad \forall e \in E_R \tag{7.26}$$

$$x_e \geq y_e, \qquad \forall e \in E_{NR} \tag{7.27}$$

$$x_e, y_e \in \{0,1\}, \qquad \forall e \in E_{NR}. \tag{7.28}$$

is a polytope in $\mathbb{Z}^{|E_R|+2|E_{NR}|}$, denoted 1–RPP$(G)$, of dimension $2|E_{NR}|$ (if graph (V, E_{NR}) is 3–edge connected) and several families of valid and facet–inducing inequalities were described.

Let us consider now the graph $G = (V, E_R \cup E_{NR})$ where the LDdGRP is defined. Given two nodes $v_i, v_f \in V_R^0$, we define the Open Rural Postman Problem (ORPP) on graph G as the problem of finding a minimum cost walk starting at vertex v_i, servicing each required edge of the graph exactly once, and ending at vertex v_f. We can build a new graph $\bar{G} = (V, \bar{E}_R \cup E_{NR})$, with $\bar{E}_R = E_R \cup \{(v_i, v_f)\}$, in such a way that solving the ORPP on graph G is equivalent to solving the 1–RPP on graph $\bar{G}$ and, then, removing a copy of edge (v_i, v_f). Note that for each ORPP solution on G we have a 1–RPP solution on $\bar{G}$, and viceversa, and therefore it can be seen that both polyhedron have the same dimension. Since (V_R^0, E_{NR}) is a complete graph, $\bar{G}$ already contains a non–required edge parallel to edge (v_i, v_f), and the dimension of 1–RPP$(\bar{G})$ is $2|E_{NR}|$. Then, the following result holds:

Theorem 7.3.3. *If graph (V, E_{NR}) is 3–edge connected, then dim*(ORPP$(G)) = 2|E_{NR}|$.

Note 7.3.3.1. Given $f(x, y) \geq \alpha$ a facet–inducing inequality of 1–RPP$(\bar{G})$, it can be seen that by replacing the variable $x_{(v_i,v_f)}$ by 1, the resulting inequality is valid and facet–inducing of ORPP(G). In particular, if graph (V, E_{NR}) is 3–edge connected, inequalities $y_e \geq 0$ and $x_e \leq 1$ are facet–inducing of ORPP(G), for each $e \in E_{NR}$, and inequalities $x_e \geq y_e$ for each $e \in E_{NR}$ are facet–inducing of ORPP(G) if graph $(V, E_{NR} \setminus \{e\})$ is 3–edge connected (see Theorems 4.2.3, 4.2.4, and 4.2.5).

Let LDdGRP(G) denotes the polytope defined as the convex hull of all the vectors

$$(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \mathbb{Z}^{P(|E_R|+2|E_{NR}|+|V_R|+1)}$$

corresponding to LDdGRP tours on G. In order to study conditions under which the inequalities deduced from the above formulation induce facets of LDdGRP(G), it is necessary to first know the dimension of the polyhedron.

Theorem 7.3.4. *If (V, E_{NR}) is a 3–edge connected graph, then*

$$dim(\text{LDdGRP}(G)) = P(|E_R| + 2|E_{NR}|) + (|V_R| - 1)^2.$$

$$
\begin{array}{c:c:c:c:c}
\text{Path 1} & \text{Path 2} & & \text{Path } P & \mathbf{z} \\
 & t_1^2 & & t_1^P & \mathbf{z}^1 \\
T^1 & \cdots & \cdots & \cdots & \cdots \\
 & t_1^2 & & t_1^P & \mathbf{z}^1 \\
\hdashline
t_1^1 & & & t_1^P & \mathbf{z}^1 \\
\cdots & T^2 & \cdots & \cdots & \cdots \\
t_1^1 & & & t_1^P & \mathbf{z}^1 \\
\hdashline
\cdots & \cdots & \cdots & \cdots & \cdots \\
\hdashline
t_1^1 & t_1^2 & & & \mathbf{z}^1 \\
\cdots & \cdots & \cdots & T^P & \cdots \\
t_1^1 & t_1^2 & & & \mathbf{z}^1 \\
\hline
t_1^1(e_1) & t_1^2 & & t_1^P & \mathbf{z}^1 \\
\cdots & \cdots & \cdots & \cdots & \cdots \\
t_1^1(e_{|E_R|}) & t_1^2 & & t_1^P & \mathbf{z}^1 \\
\hdashline
t_1^1 & t_1^2(e_1) & & t_1^P & \mathbf{z}^1 \\
\cdots & \cdots & \cdots & \cdots & \cdots \\
t_1^1 & t_1^2(e_{|E_R|}) & & t_1^P & \mathbf{z}^1 \\
\hdashline
\cdots & \cdots & \cdots & \cdots & \cdots \\
\hdashline
t_1^1 & t_1^2 & & t_1^P(e_1) & \mathbf{z}^1 \\
\cdots & \cdots & \cdots & \cdots & \cdots \\
t_1^1 & t_1^2 & & t_1^P(e_{|E_R|}) & \mathbf{z}^1 \\
\hline
 & & & & \mathbf{z}^2 \\
* & * & \cdots & * & \cdots \\
 & & & & \mathbf{z}^{w+1}
\end{array}
$$

Figure 7.2: Matrix of LDdGRP solutions appearing in the proof of Theorem 7.3.4

Proof: Since all the points of LDdGRP(G) satisfy $4|V_R|$ linearly independent equations (Lemma 7.3.1), we have that $dim(\text{LDdGRP}(G)) \leq P(|E_R|+2|E_{NR}|+|V_R|+1)-4|V_R| = P(|E_R|+2|E_{NR}|)+(|V_R|-1)^2$. In order to prove that $dim(\text{LDdGRP}(G)) \geq P(|E_R|+2|E_{NR}|)+(|V_R|-1)^2$, we have to find $P(|E_R|+2|E_{NR}|)+(|V_R|-1)^2+1$ affinely independent LDdGRP tours.

From Theorem 7.3.2, there are $w+1$ affinely independent vectors $\mathbf{z}^1, \mathbf{z}^2, \ldots, \mathbf{z}^{w+1}$, where $w = (|V_R|-1)^2$. In order to obtain LDdGRP tours, we have to complete these vectors $\mathbf{z}$ with vectors $(\mathbf{x}, \mathbf{y}) \in \mathbb{Z}^{P(|E_R|+2|E_{NR}|)}$.

Let us consider $\mathbf{z}^1$ one of the previous vectors. This vector provides an order in which vertices in V_R are serviced in the corresponding LDdGRP solution: each path p, with $p = 1, \ldots, P$, starts at the vertex $u \in V_R^0$ such that $(\mathbf{z}^1)_u^{p-1} = 1$ and ends at the vertex $v \in V_R^0$ such that $(\mathbf{z}^1)_v^p = 1$. Consider the ORPP defined on graph G with initial vertex $v_i = u$ and final vertex $v_f = v$. From Theorem 7.3.3, and since (V, E_{NR}) is 3–edge connected, there are $m+1$ vectors $t^p = (x^p, y^p)$ affinely independent, say $t_1^p, \ldots, t_{m+1}^p$, where $m = 2|E_{NR}|$, each of them starting at u, ending at v, and satisfying $x_i(e) = 1$ for all the edges in E_R. Note that $(t_{j_1}^1, \ldots, t_{j_P}^P, \mathbf{z}^1)$ is an LDdGRP solution,

$$\begin{array}{c:c:c:c:c}
\text{Path 1} & \text{Path 2} & & \text{Path } P & \mathbf{z} \\
T^1 - t_1^1 & 0 & \cdots & 0 & 0 \\ \hdashline
0 & T^2 - t_1^2 & \cdots & 0 & 0 \\ \hdashline
\cdots & \cdots & \cdots & \cdots & \cdots \\ \hdashline
0 & 0 & \cdots & T^P - t_1^P & 0 \\ \hline
\begin{matrix} -1-1 & & \\ & \ddots & \\ & & -1-1 \end{matrix} & 0 & \cdots & 0 & 0 \\ \hdashline
0 & \begin{matrix} -1-1 & & \\ & \ddots & \\ & & -1-1 \end{matrix} & \cdots & 0 & 0 \\ \hdashline
\cdots & \cdots & \cdots & \cdots & \cdots \\ \hdashline
0 & 0 & \cdots & \begin{matrix} -1-1 & & \\ & \ddots & \\ & & -1-1 \end{matrix} & 0 \\ \hline
* & * & \cdots & * & \begin{matrix} \mathbf{z}^2 - \mathbf{z}^1 \\ \cdots \\ \mathbf{z}^{w+1} - \mathbf{z}^1 \end{matrix}
\end{array}$$

Figure 7.3: Matrix appearing in the proof of Theorem 7.3.4

for all $j_1, \ldots, j_P \in \{1, 2, \ldots, m+1\}$. It can be assumed that each t_1^p, for each path p, is formed with two copies of each edge in E_{NR} incident with vertices in $V_I \cup \{u, v\}$, and then replacing one copy of each $e \in E'_{NR}$ by the required edge parallel to e and removing a copy of (u, v). For each required $e \in E_R$, we can build a vector $t_1^p(e)$ from t_1^p by removing the traversal of edges e and e', where e' denotes the non–required edge parallel to e. Note that, for example, $(t_1^1(e_1), t_{j_2}^2, \ldots, t_{j_P}^P, \mathbf{z}^1)$, is also an LDdGRP solution.

We can build $P(m+1)$ LDdGRP solutions in the following way. For each $i \in \{1, \ldots, P\}$, let T^i be the matrix formed by all the affinely independent ORPP solutions $t_1^i, \ldots, t_{m+1}^i$ as rows. Path $p = i$ performs any ORPP walk t_j^i above, while the other paths perform t_1^p, $p \neq i$. These solutions are depicted as the rows of the P first block rows in the matrix shown in Figure 7.2. Note that the first row of each block rows are equal to $(t_1^1, t_1^2, \ldots, t_1^P, \mathbf{z}^1)$. Furthermore, we can build $P|E_R|$ more LDdGRP solutions as follows. For each $i \in \{1, \ldots, P\}$ and for each required edge $e_j \in E_R = \{e_1, \ldots, e_{|E_R|}\}$, path $p = i$ performs $t_1^i(e_j)$ while the other paths perform t_1^p, $p \neq i$. These solutions are depicted as the rows of the next P block rows in the matrix of Figure 7.2. The last block row of Figure 7.2 is formed by w LDdGRP solutions, each generated from a different vector $\mathbf{z}^2, \ldots, \mathbf{z}^P$.

After substracting the first row from all the other rows (and then removing the null ones), we obtain the matrix in Figure 7.3. A big zero in a block of the matrix means that all the entries of this block are zero. Each $T^j - t_1^j$ block has full rank because the vectors $t_1^j, \dots, t_{m+1}^j$ are affinely independent. Furthermore, in the first P block rows, all the $t_j^p - t_1^p$ vectors take value zero in all the entries corresponding to the required edges, whereas the first -1 entry in each pair $-1, -1$ in the rows of the next P block rows is associated with each edge $e \in E_R$, and then it is the only non–zero entry in the column corresponding to e. Hence, the first $P(m + |E_R|)$ rows of the matrix are linearly independent. The bottom–right block of the matrix has full rank because vectors $\mathbf{z}^1$, …, $\mathbf{z}^P$ are affinely independent, and the matrix in Figure 7.3 is a full rank matrix with $P(m + |E_R|) + w$ linearly independent rows. Therefore, we have found $P(|E_R| + 2|E_{NR}|) + (|V_R| - 1)^2 + 1$ affinely independent LDdGRP solutions, and we are done. □

In what follows, we will assume that (V, E_{NR}) is a 3–edge connected graph and, hence, $dim(\text{LDdGRP}(G)) = P(|E_R| + 2|E_{NR}|) + (|V_R| - 1)^2$.

7.3.1 Facet–inducing inequalities from the formulation

Let us prove in this section that some inequalities from the formulation induce facets of the LDdGRP polyhedron.

Theorem 7.3.5. *Inequality $x_e^p \geq 0$, for each $e \in E_R$ and for each path $p \in \{1, \dots, P\}$, is facet–inducing for* LDdGRP(G).

Proof: We will prove it for $p = 1$ and for a required edge e_1, where $E_R = \{e_1, \dots, e_{|E_R|}\}$. We have to find $dim(\text{LDdGRP}(G))$ affinely independent LDdGRP solutions satisfying $x_{e_1}^1 = 0$. Let $\mathbf{z}^1, \mathbf{z}^2, \dots, \mathbf{z}^{w+1}$ be the $w + 1$ affinely independent vectors from Theorem 7.3.2 and let us consider the vector $\mathbf{z}^1$. Recall that this vector provides an order in which vertices in V_R are serviced in the corresponding LDdGRP solution. We assume that path $p = 1$ starts at vertex $u \in V_R^0$ and ends at vertex $v \in V_R^0$. Consider the ORPP defined on graph $G \setminus \{e_1\}$ with initial vertex $v_i = u$ and final vertex $v_f = v$. From Theorem 7.3.3, and since (V, E_{NR}) is 3–edge connected, there are $m + 1$ affinely independent ORPP vectors on $G \setminus \{e_1\}$, with $m = 2|E_{NR}|$. By adding to those vectors a new component $x_{e_1} = 0$, we obtain $m + 1$ affinely independent vectors $\bar{t}_1^1, \dots, \bar{t}_{m+1}^1$ starting at u, ending at v and traversing all the required edges of G but e_1, thus satisfying $x_{e_1}^1 = 0$. It can be assumed that $\bar{t}_1^1$ is formed with two copies of each edge in E_{NR} incident with vertices in $V_I \cup \{u, v\}$, and then replacing one copy of each $e \in E'_{NR}, e \neq e'_1$, by the required edge parallel to e and removing a copy of (u, v). For each required $e \in E_R \setminus \{e_1\}$, we can build a vector $\bar{t}_1^1(e)$ from $\bar{t}_1^1$ by removing the edges e and e'.

On the other hand, let $t_i^2, \dots, t_i^P$, for $i = 1, \dots, m + 1$, and $t_1^2(e_j), \dots, t_1^P(e_j)$, for $e_j \in E_R$, be the vectors built as in Theorem 7.3.4. With all these vectors, we can build LDdGRP solutions as in the proof of Theorem 7.3.4 that can be arranged in a matrix with $(m + 1)P + |E_R|P - 1 + w$ rows similar to the matrix in Figure 7.2. After substracting the first row from all the other rows (and then removing the null ones), we obtain a full–rank matrix with $mP + |E_R|P - 1 + w$ rows and, hence, we have $P(m + |E_R|) + w = dim(\text{LDdGRP}(G))$ affinely independent LDdGRP tours satisfying $x_{e_1}^1 = 0$, and we are done. □

Theorem 7.3.6. *Inequality $x_e^p \leq 1$, for each edge $e \in E_R \cup E'_{NR}$ and for each path $p \in \{1, \ldots, P\}$, is facet–inducing for* LDdGRP(G).

Proof: We will prove it for $p = 1$. For a required edge e_1, where $E_R = \{e_1, \ldots, e_{|E_R|}\}$, all the LDdGRP solutions in the matrix of Figure 7.2 but one of them $(t_1^1(e_1))$ satisfy $x_{e_1}^1 = 1$, and we are done.

For an edge $e'_1 \in E'_{NR}$, the non–required edge parallel to a required edge e_1, all the LDdGRP solutions in the matrix of Figure 7.2 satisfy $x_{e'_1}^1 = 1$ except $t_1^1(e_1)$ and some rows of the matrix block T^1. Recall that all the rows in the matrix of Figure 7.2 satisfy $x_{e_1}^1 = 1$ except the one corresponding to $t_1^1(e_1)$. If we permute in that matrix the values of the columns corresponding to the $x_{e_1}^1$ and $x_{e'_1}^1$ variables, we obtain a new matrix with the same rank as before and with all the rows (but one) satisfying $x_{e'_1}^1 = 1$. Note that all the rows of this matrix are LDdGRP solutions because each path $p \geq 2$ services all the required edges. Hence, we have dim(LDdGRP(G)) affinely independent LDdGRP solutions satisfying $x_{e'_1}^1 = 1$. □

Theorem 7.3.7. *Inequality $x_e^p \leq 1$, for each edge $e \in E''_{NR}$ and for each path $p \in \{1, \ldots, P\}$, is facet–inducing for* LDdGRP(G) *if $(V, E_{NR} \backslash \{e\})$ is a 3–edge connected graph.*

Proof: We will prove it for $p = 1$. Since $(V, E_{NR} \setminus \{e\})$ is 3–edge connected, we have $dim(\text{ORPP}(G \setminus \{e\})) = 2(|E_{NR}| - 1) = m - 2$, and there are $m - 1$ affinely independent ORPP solutions on graph $G \backslash \{e\}$. Let T^* be the corresponding incidence matrix of these vectors expressed as rows. We can assume that t_1^1 is the path as defined in the proof of Theorem 7.3.4. We complete these solutions with $x_e^1 = y_e^1 = 1$ and we obtain $m - 1$ ORPP solutions in G satisfying $x_e^1 = 1$. Moreover, we can build another ORPP solution on G by completing t_1^1 with $x_e^1 = 1$ and $y_e^1 = 0$, and deleting the second traversal of the edges in a path p_{ij} joining the two endpoints of $e = (i, j)$. This last ORPP solution also satisfies $x_e^1 = 1$. Let $\bar{T}^1$ be the incidence matrix of these m ORPP solutions on G expressed as rows, depicted in Figure 7.4a. By substracting the first row from all the other rows, we obtain the matrix in Figure 7.4b, whose $m - 1$ non–zero rows are linearly independent. As in the previous proofs, we can build a matrix as the one depicted in Figure 7.2 with $\bar{T}^1$ instead of T^1. After substracting the first row from all the other rows, we obtain a matrix similar to that in Figure 7.3 with $m - 1 + m(P - 1) + |E_R|P + w$ linearly independent rows, and we are done. □

Theorem 7.3.8. *Inequality $y_e^p \geq 0$, for each $e \in E'_{NR}$ and for each path $p \in \{1, \ldots, P\}$, is facet–inducing for* LDdGRP(G).

Proof: We will prove it for $p = 1$ and for a non–required edge e'_1 parallel to e_1, where $E_R = \{e_1, \ldots, e_{|E_R|}\}$. We have to find dim(LDdGRP(G)) affinely independent LDdGRP solutions satisfying $y_{e'_1}^1 = 0$. Let $\mathbf{z}^1, \mathbf{z}^2, \ldots, \mathbf{z}^{w+1}$ be the w+1 affinely independent vectors from Theorem 7.3.2 and let us consider the vector $\mathbf{z}^1$, that provides an order in which vertices in V_R are serviced in the corresponding LDdGRP solution. We assume that path $p = 1$ starts at vertex $u \in V_R^0$ and ends at vertex $v \in V_R^0$. Consider the ORPP defined on graph G with initial vertex $v_i = u$ and final vertex $v_f = v$. From Note 7.3.3.1,

$$\begin{array}{c:cc} & x_e^1 & y_e^1 \\ t_1^1 & 1 & 1 \\ T^* & \multicolumn{2}{c}{\cdots} \\ & 1 & 1 \\ \hdashline t_1^1 - p_{ij} & 1 & 0 \end{array} \qquad \begin{array}{c:cc} & x_e^1 & y_e^1 \\ & 0 & 0 \\ T^* - t_1^1 & \multicolumn{2}{c}{\cdots} \\ & 0 & 0 \\ \hdashline p_{ij} & 0 & 1 \end{array}$$

(a) (b)

Figure 7.4: Matrices appearing in the proof of Theorem 7.3.7

inequality $y_{e'_1} \geq 0$ is facet–inducing of ORPP(G) and, therefore, there are m affinely independent ORPP vectors on G satisfying $y_{e'_1} = 0$, say $\bar{t}_1^1, \bar{t}_2^1, \ldots, \bar{t}_m^1$. We assume that $\bar{t}_1^1$ is built as in Theorem 7.3.4. For each required $e \in E_R$, we can build a vector $\bar{t}_1^1(e)$ from $\bar{t}_1^1$ by removing the traversal of edges e and e'.

On the other hand, let $t_i^2, \ldots, t_i^P$, for $i = 1, \ldots, m+1$, and $t_1^2(e), \ldots, t_1^P(e)$, for $e \in E_R$, be the vectors built as in Theorem 7.3.4. With all these vectors $\bar{t}$ and t, we can build LDdGRP solutions as in the proof of Theorem 7.3.4 satisfying $y^1_{e'_1} = 0$ that can be arranged in a matrix with $(m+1)P - 1 + |E_R|P + w$ rows similar to the matrix in Figure 7.2. After substracting the first row from all the other rows (and then removing the null ones), we obtain a full–rank matrix with $mP - 1 + |E_R|P + w$ rows and, hence, we have dim(LDdGRP(G)) affinely independent LDdGRP tours satisfying $y^1_{e'_1} = 0$, and we are done. □

Theorem 7.3.9. *Inequality $y_e^p \geq 0$, for each $e \in E''_{NR}$ and for each path $p \in \{1, \ldots, P\}$, is facet–inducing for* LDdGRP(G) *if $(V, E_{NR} \setminus \{e\})$ is a 3–edge connected graph.*

Proof: We will prove it for $p = 1$. Since $(V, E_{NR} \setminus \{e\})$ is 3–edge connected, we have $dim(\text{ORPP}(G \setminus \{e\})) = 2(|E_{NR}| - 1) = m - 2$, and there are $m - 1$ affinely independent ORPP solutions on graph $G \setminus \{e\}$. Let T^* be the corresponding incidence matrix of these vectors expressed as rows. We can assume that t_1^1 is the path as defined in the proof of Theorem 7.3.4. We complete these solutions with $x_e^1 = y_e^1 = 0$ and we obtain $m - 1$ ORPP solutions in G satisfying $y_e^1 = 0$. Moreover, we can build another ORPP solution on G by completing t_1^1 with $x_e^1 = 1$, $y_e^1 = 0$, and deleting a traversal of the edges in a path p_{ij} joining the two endpoints of $e = (i, j)$. This last ORPP solution also satisfies $y_e^1 = 0$. The remaining of the proof is similar to that of Theorem 7.3.7. □

Theorem 7.3.10. *Inequality $x_e^p \geq y_e^p$, for each $e \in E'_{NR}$ and for each path $p \in \{1, \ldots, P\}$, is facet–inducing for* LDdGRP(G) *if graph $(V, E_{NR} \setminus \{e\})$ is 3–edge connected.*

Proof: We will prove it for $p = 1$ and for a non–required edge e'_1 parallel to e_1, where $E_R = \{e_1, \ldots, e_{|E_R|}\}$. Let us consider the graph $\bar{G} = G \setminus \{e_1\}$. Note that, in $\bar{G}$, edge e'_1 is now in E''_{NR}. Given that $(V, E_{NR} \setminus \{e'_1\})$ is 3–edge connected, inequality $x_{e'_1} \geq y_{e'_1}$

is facet–inducing of ORPP($\bar{G}$) and there are m affinely independent ORPP solutions in $\bar{G}$ satisfying $x_{e'_1} = y_{e'_1}$, say $\bar{t}^1_1, \bar{t}^1_2, \ldots, \bar{t}^1_m$. We can assume that $\bar{t}^1_1$ is built in $\bar{G}$ as in Theorem 7.3.4. If we complete these vectors with a component $x_{e_1} = 0$, we obtain m affinely independent ORPP solutions in G. For each required edge $e \in E_R$, with $e \neq e_1$, we can build a vector $\bar{t}^1_1(e)$ from $\bar{t}^1_1$ by removing the edges e and e'. For the required edge e_1, we can build the vector $\bar{t}^1_1(e_1)$ from $\bar{t}^1_1$ by adding the edge e_1 and removing a traversal of the non–required edges in a path p_{ij} joining the two endpoints of e_1. The remaining of the proof is similar to that of Theorem 7.3.7. □

Theorem 7.3.11. *Inequality $x^p_e \geq y^p_e$, for each $e \in E''_{NR}$ and for each path $p \in \{1, \ldots, P\}$, is facet–inducing for* LDdGRP(G) *if graph $(V, E_{NR} \setminus \{e\})$ is 3–edge connected.*

Proof: We will prove it for $p = 1$ and for a non–required edge $e \in E''_{NR}$. We have to find dim(LDdGRP(G)) affinely independent LDdGRP solutions satisfying $x^1_e = y^1_e$. Let $\mathbf{z}^1, \mathbf{z}^2, \ldots, \mathbf{z}^{w+1}$ be the $w + 1$ affinely independent vectors from Theorem 7.3.2 and let us consider the vector $\mathbf{z}^1$, that provides an order in which vertices in V_R are serviced in the corresponding LDGRP solution. We assume that path $p = 1$ starts at vertex $u \in V^0_R$ and ends at vertex $v \in V^0_R$. Consider the ORPP defined on graph G with initial vertex $v_i = u$ and final vertex $v_f = v$. From Note 7.3.3.1, and given that graph $(V, E_{NR} \setminus \{e\})$ is 3–edge connected, inequality $x_e \geq y_e$ is facet–inducing of ORPP(G) and therefore there are m affinely independent ORPP vectors on G satisfying $x_e = y_e$, say $\bar{t}^1_1, \bar{t}^1_2, \ldots, \bar{t}^1_m$. We can assume that $\bar{t}^1_1$ is built as in Theorem 7.3.4, which satisfies $x_e = y_e = 1$. For each required $e \in E_R$, we can build a vector $\bar{t}^1_1(e)$ from $\bar{t}^1_1$ by removing the edges e and e'. The remaining of the proof is similar to that of Theorem 7.3.8. □

Note 7.3.11.1. *Inequalities $x_e \geq 0$ and $y_e \leq 1$, for each $e \in E_{NR}$, are not facet–inducing of* LDdGRP(G) *since they are dominated by inequalities $x_e \geq y_e$.*

Theorem 7.3.12. *Inequality $\sum_{p=1}^{P} x^p_e \geq 1$, for each $e \in E_R$, is facet–inducing for* LDdGRP(G).

Proof: For a required edge $e_1 \in E_R$, where $E_R = \{e_1, \ldots, e_{|E_R|}\}$, we have to find dim(LDdGRP(G)) affinely independent LDdGRP solutions satisfying $\sum_{p=1}^{P} x^p_{e_1} = 1$. Let $\mathbf{z}^1, \mathbf{z}^2, \ldots, \mathbf{z}^{w+1}$ be the $w+1$ affinely independent vectors $\mathbf{z}$ from Theorem 7.3.2 and let us consider the vector $\mathbf{z}^1$, that provides an order in which vertices in V_R are serviced in the corresponding LDdGRP solution.

For each path $p \in \{1, \ldots, P\}$, we can consider an ORPP defined on graph G starting and ending at the corresponding vertices provided by $\mathbf{z}^1$. From Theorem 7.3.3, and since (V, E_{NR}) is 3–edge connected, there are $m + 1$ affinely independent ORPP vectors, say $t^p_1, \ldots, t^p_{m+1}$, where $m = 2|E_{NR}|$, satisfying $x^p_{e_1} = 1$. It can be assumed that the first of them, t^p_1, is defined as in the proof of Theorem 7.3.4, which satisfies $x^p_{e_1} = x^p_{e'_1} = 1$ and $y^p_{e'_1} = 0$. We can build a vector $\bar{t}^p_1$ from t^p_1 by replacing the traversal of edge e_1 by a second traversal of the corresponding parallel edge e'_1, thus satisfying $x^p_{e'_1} = y^p_{e'_1} = 1$

$$
\begin{array}{c}
\begin{array}{ccccc}
\text{Path } 1 & \text{Path } 2 & & \text{Path } P & \mathbf{z}
\end{array}\\
\left(\begin{array}{c:c:c:c:c}
 & \bar{t}_1^2 & & \bar{t}_1^P & \mathbf{z}^1\\
T^1 & \cdots & \cdots & \cdots & \cdots\\
 & \bar{t}_1^2 & & \bar{t}_1^P & \mathbf{z}^1\\
\hdashline
\bar{t}_1^1 & & & \bar{t}_1^P & \mathbf{z}^1\\
\cdots & T^2 & \cdots & \cdots & \cdots\\
\bar{t}_1^1 & & & \bar{t}_1^P & \mathbf{z}^1\\
\hdashline
\cdots & \cdots & \cdots & \cdots & \cdots\\
\hdashline
\bar{t}_1^1 & \bar{t}_1^2 & & & \mathbf{z}^1\\
\cdots & \cdots & \cdots & T^P & \cdots\\
\bar{t}_1^1 & \bar{t}_1^2 & & & \mathbf{z}^1\\
\hline
t_1^1(e_2) & \bar{t}_1^2 & & \bar{t}_1^P & \mathbf{z}^1\\
\cdots & \cdots & \cdots & \cdots & \cdots\\
t_1^1(e_{|E_R|}) & \bar{t}_1^2 & & \bar{t}_1^P & \mathbf{z}^1\\
\hdashline
t_1^1 & t_1^2(e_1) & & \bar{t}_1^P & \mathbf{z}^1\\
\bar{t}_1^1 & t_1^2(e_2) & \cdots & \bar{t}_1^P & \mathbf{z}^1\\
\cdots & \cdots & & \cdots & \cdots\\
\bar{t}_1^1 & t_1^2(e_{|E_R|}) & & \bar{t}_1^P & \mathbf{z}^1\\
\hdashline
\cdots & \cdots & \cdots & \cdots & \cdots\\
\hdashline
t_1^1 & \bar{t}_1^2 & & t_1^P(e_1) & \mathbf{z}^1\\
\bar{t}_1^1 & \bar{t}_1^2 & \cdots & t_1^P(e_2) & \mathbf{z}^1\\
\cdots & \cdots & & \cdots & \cdots\\
\bar{t}_1^1 & \bar{t}_1^2 & & t_1^P(e_{|E_R|}) & \mathbf{z}^1\\
\hline
 & & & & \mathbf{z}^2\\
* & * & \cdots & * & \cdots\\
 & & & & \mathbf{z}^{w+1}
\end{array}\right)
\end{array}
$$

Figure 7.5: Matrix of LDdGRP solutions appearing in the proof of Theorem 7.3.12

and $x_{e_1}^p = 0$. Moreover, for each required $e \in E_R$, $e \neq e_1$, we can build a vector $t_1^p(e)$ from t_1^p by removing the traversal of the edge e and its corresponding parallel e'.

We can build $P(m+1)$ LDdGRP solutions in the following way. As in Theorem 7.3.4, let T^i be the matrix formed by all the affinely independent ORPP solutions $t_1^i, \dots, t_{m+1}^i$ as rows, for each path $i \in \{1, \dots, P\}$. A path i performs any tour t_j^i from T^i, while the other paths perform $\bar{t}_1^p$, $p \neq i$. These $P(m+1)$ LDdGRP solutions satisfy $\sum_{p=1}^P x_{e_1}^p = 1$ and can be depicted as the rows of the P first block rows in the matrix in Figure 7.5. The block row $P+1$ of this matrix contains the solutions in which the first path performs $t_1^1(e_j)$, for each $j = 2, \dots, |E_R|$, while the other paths perform $\bar{t}_1^p$. The block row $P+2$ contains a first solution in which the first path performs t_1^1, the second path performs $t_1^2(e_1)$, and the others paths perform $\bar{t}_1^p$, and $|E_R| - 1$ solutions more in which path 2 performs a vector $t_1^1(e_j)$, with $j = 2, \dots, |E_R|$, while the others perform $\bar{t}_1^p$, for $p \neq 2$. Block rows $P+3, \dots, 2P$ are built in a similar way. Note that all these LDdGRP solutions also satisfy $\sum_{p=1}^P x_{e_1}^p = 1$. A last block row formed by w LDdGRP

$$
\begin{array}{c:c:c:c:c}
\text{Path 1} & \text{Path 2} & & \text{Path } P & \mathbf{z} \\
T^1 - t_1^1 & 0 & \cdots & 0 & 0 \\ \hdashline
\bar{t}_1^1 - t_1^1 & T^2 - \bar{t}_1^2 & \cdots & 0 & 0 \\ \hdashline
\cdots & \cdots & \cdots & \cdots & \cdots \\ \hdashline
\bar{t}_1^1 - t_1^1 & 0 & \cdots & T^P - \bar{t}_1^P & 0 \\ \hline
\begin{matrix} -1-1 & & \\ & \ddots & \\ & & -1-1 \end{matrix} & 0 & \cdots & 0 & 0 \\ \hdashline
\bar{t}_1^1 - t_1^1 & \begin{matrix} -1-1 & & \\ & \ddots & \\ & & -1-1 \end{matrix} & \cdots & 0 & 0 \\ \hdashline
\cdots & \cdots & \cdots & \cdots & \cdots \\ \hdashline
\bar{t}_1^1 - t_1^1 & 0 & \cdots & \begin{matrix} -1-1 & & \\ & \ddots & \\ & & -1-1 \end{matrix} & 0 \\ \hline
* & * & \cdots & * & \begin{matrix} \mathbf{z}^2 - \mathbf{z}^1 \\ \cdots \\ \mathbf{z}^{w+1} - \mathbf{z}^1 \end{matrix}
\end{array}
$$

Figure 7.6: Matrix appearing in the proof of Theorem 7.3.12

solutions, each generated from a different vector $\mathbf{z}^2, \ldots, \mathbf{z}^P$, is added to the matrix in Figure 7.5.

If we subtract the first row from all the other rows in the matrix in Figure 7.5, we obtain the matrix in Figure 7.6. The top–left block of this matrix contains m linearly independent rows. The next $P-1$ diagonal blocks, $T^i - \bar{t}_1^i$, contain $m+1$ affinely independent rows, since the $m+1$ rows of T^i are affinely independent, and then there are at least m linearly independent rows on each of these blocks. Therefore, there are at least Pm linearly independent rows in the first P block rows of the matrix. Furthermore, each row in the next P block rows contains an entry with value -1 associated with a variable $x^p_{e'_j}$, for some path p and $j \in \{1, \ldots, |E_R|\}$, and this entry is the only one in its column taking non–zero value. Hence, the $P|E_R| - 1$ rows in this blocks are linearly independent. In addition, the bottom–right block of the matrix has full range because vectors $\mathbf{z}^1, \ldots, \mathbf{z}^P$ are affinely independent. Therefore, we have $P(m + |E_R|) - 1 + (|V_R|^2 - 1)^2$ linearly independent rows in the matrix in Figure 7.5, we have found $P(|E_R| + 2|E_{NR}|) + (|V_R| - 1)^2$ affinely independent LDdGRP solutions satisfying $\sum_{p=1}^{P} x^p_{e_1} = 1$, and we are done. $\square$

Theorem 7.3.13. *Connectivity inequalities* (7.4),

$$x^p(\delta_R(S)) + (x^p + y^p)(\delta_{NR}(S)) \geq 2x_f^p - \sum_{i\in S\cap V_R^0} \left(z_i^{p-1} + z_i^p\right),$$

$\forall S \subset V$, $\forall f \in E(S)$, $\forall p \in \{1,\dots,P\}$, *are facet–inducing for* LDdGRP(G) *if graphs* $G(S)$ *and* $G(V\setminus S)$ *are 3–edge connected and* $|(V\setminus S)\cap V_R^0| \geq 2$.

Proof: We will prove it for $p=1$. Note that if $(V\setminus S)\cap V_R^0 = \emptyset$, then $\sum_{i\in S\cap V_R^0}(z_i^{p-1}+z_i^p) = 2$ holds, and the inequality (7.4) is dominated by $x^p(\delta_R(S)) + (x^p+y^p)(\delta_{NR}(S)) \geq 0$, which can not induce a facet.

In some points of this proof we will distinguish the cases $f \in E_{NR}$ and $f \in E_R$. We have to find dim(LDdGRP(G)) affinely independent LDdGRP solutions satisfying inequality (7.4) as an equality. Let $\mathbf{z}^1, \mathbf{z}^2, \dots, \mathbf{z}^{w+1}$ be the $w+1$ affinely independent vectors from Theorem 7.3.2 and let us consider the vector $\mathbf{z}^1$, that provides an order in which vertices in V_R are serviced in the corresponding LDdGRP solution. Since $|(V\setminus S)\cap V_R^0| \geq 2$, we can assume that path $p=1$ starts at vertex $u \in (V\setminus S)\cap V_R^0$ and ends at vertex $v \in (V\setminus S)\cap V_R^0$. Consider the graph G^* obtained by deleting from G the required edges in $\delta_R(S)\cup E_R(S)$ and consider the ORPP defined on G^* with initial vertex $v_i = u$ and final vertex $v_f = v$. From Note 7.3.3.1 and Theorem 4.2.6, and given that $E_R(S) = \delta_R(S) = \emptyset$ in G^*, and graphs $G^*(S)$ and $G^*(V\setminus S)$ are 3–edge connected, inequality $x(\delta_R(S)) + (x+y)(\delta_{NR}(S)) \geq 2x_{\bar f}$ is facet–inducing of ORPP(G^*), where, if edge $f \in E_R$, $\bar f = f'$, and $\bar f = f$ otherwise. Therefore, as graphs G and G^* have the same set of non–required edges, there are $m = 2|E_{NR}|$ affinely independent ORPP vectors $\bar t_1^*, \bar t_2^*, \dots, \bar t_m^*$ on G^*, all of them traversing the required edges $a \in E_R(V\setminus S)$ and satisfying $x(\delta_R(S)) + (x+y)(\delta_{NR}(S)) = 2x_{\bar f}$. Each $\bar t_j^*$ is transformed into an ORPP vector on G, say $\bar t_j^1$, by adding to it an extra entry x_e with value $x_e = 0$ for each previously removed edge $e \in \delta_R(S)\cup E_R(S)$. In the case $f \in E_R$, we replace a traversal of f' (if any) by the traversal of f. It can be seen that the resulting ORPP vectors on G, $\bar t_1^1, \bar t_2^1, \dots, \bar t_m^1$, are also affinely independent, satisfy $x(\delta_R(S)) + (x+y)(\delta_{NR}(S)) = 2x_f$, traverse all the required edges in $E_R(V\setminus S)$, and do not traverse any edge in $\delta_R(S)\cup E_R(S)$ (except f if it is required). By the consideration about vector $\mathbf{z}^1$, vectors $(\bar t_j^1, \mathbf{z}^1)$, $j = 1,\dots,m$, satisfy inequality (7.4) as an equality.

We can assume that one of these ORPP vectors, say $\bar t_1^1$, traverses all the edges in $E(V\setminus S)$ twice, traverses the cutset $\delta(S)$ twice through a non–required edge, say $\bar e$, and satisfies $x_1(a) = y_1(a) = 1$ for all $a \in E_{NR}(S)$, $x_1(a) = 0$ for all $a \in E_R(S)$, in the case $f \in E_{NR}$, and $x_1(f) = x_1(f') = 1$, $y_1(f') = 0$, and $x_1(a) = 0$ for all $a \in E_R(S)\setminus\{f\}$, in the case $f \in E_R$.

On the other hand, let $t_i^2, \dots, t_i^P$, for $i = 1,\dots,m+1$, and $t_1^2(e), \dots, t_1^P(e)$, for $e \in E_R$, be the vectors built as in Theorem 7.3.4. With all these vectors $\bar t$ and t, we can build LDdGRP solutions satisfying inequality (7.4) as an equality: path $p=1$ performs any ORPP vector $\bar t_j^1$ above, while the other paths $p>1$ perform t_i^p. These LDdGRP solutions can be arranged in a matrix whose first P block rows are similar to the matrix in Figure 7.5 but using vectors $\bar t$ in path $p=1$.

For the path $p=1$ and for each required edge in $\{e_1, e_2, \dots, e_{|E_R|}\}$, we define the vectors $\bar t_1^1(e_i)$ as follows. For each required edge $a \in E_R(V\setminus S)$, we define the vector

$$
\begin{array}{c|c|c|c|c}
\text{Path 1} & \text{Path 2} & & \text{Path } P & \mathbf{z} \\
\hline
 & t_1^2 & & t_1^P & \mathbf{z}^1 \\
\bar{T}^1 & \dots & \dots & \dots & \dots \\
 & t_1^2 & & t_1^P & \mathbf{z}^1 \\
\hline
\bar{t}_1^1 & & & t_1^P & \mathbf{z}^1 \\
\dots & T^2 & \dots & \dots & \dots \\
\bar{t}_1^1 & & & t_1^P & \mathbf{z}^1 \\
\hline
\dots & \dots & \dots & \dots & \dots \\
\hline
\bar{t}_1^1 & t_1^2 & & & \mathbf{z}^1 \\
\dots & \dots & \dots & T^P & \dots \\
\bar{t}_1^1 & t_1^2 & & & \mathbf{z}^1 \\
\hline
\bar{t}_1^1(e_1) & t_1^2 & & t_1^P & \mathbf{z}^1 \\
\dots & \dots & \dots & \dots & \dots \\
\bar{t}_1^1(e_{|E_R|}) & t_1^2 & & t_1^P & \mathbf{z}^1 \\
\hline
\bar{t}_1^1 & t_1^2(e_1) & & t_1^P & \mathbf{z}^1 \\
\dots & \dots & \dots & \dots & \dots \\
\bar{t}_1^1 & t_1^2(e_{|E_R|}) & & t_1^P & \mathbf{z}^1 \\
\hline
\dots & \dots & \dots & \dots & \dots \\
\hline
\bar{t}_1^1 & t_1^2 & & t_1^P(e_1) & \mathbf{z}^1 \\
\dots & \dots & \dots & \dots & \dots \\
\bar{t}_1^1 & t_1^2 & & t_1^P(e_{|E_R|}) & \mathbf{z}^1 \\
\hline
 & & & & \mathbf{z}^2 \\
* & * & \dots & * & \dots \\
 & & & & \mathbf{z}^{w+1}
\end{array}
$$

Figure 7.7: Matrix of LDdGRP solutions appearing in the proof of Theorem 7.3.13

$\bar{t}_1^1(a)$ equal to $\bar{t}_1^1$ after removing the traversal of the edges a and a'. For each required edge $a \in \delta_R(S)$, we define the vector $\bar{t}_1^1(a)$ equal to $\bar{t}_1^1$ after removing the two traversals of edge $\bar{e}$ and adding the traversals of edges a and a'. Note that if $\bar{e} = a'$ then $\bar{t}_1^1(a)$ is equal to $\bar{t}_1^1$ after replacing the second traversal of $\bar{e}$ by a traversal of edge a.

In the case $f \in E_{NR}$, for each required edge $a \in E_R(S)$, we define the vector $\bar{t}_1^1(a)$ equal to $\bar{t}_1^1$ after replacing the second traversal of a' by the traversal of a. In the case $f \in E_R$, for each $a \in E_R(S)\setminus\{f\}$, we define the vectors $\bar{t}_1^1(a)$ as before. Moreover, for edge f we define the vector $\bar{t}_1^1(f)$ equal to $\bar{t}_1^1$ in those entries corresponding to edges in $E(V\setminus S)$, and zero in the remaining ones. Note that this vector also satisfies $x(\delta_R(S)) + (x+y)(\delta_{NR}(S)) = 2x_f = 0$.

If $P \geq 3$, using these vectors $\bar{t}_i^j, \bar{t}_1^1(a), t_i^j$ and $t_1^1(a)$, and also $\mathbf{z}^1$, we can built the LDdGRP solutions represented in the following P block rows of the matrix in Figure 7.7, which satisfy inequality (7.4) as an equality. If we subtract the first row from all the other rows and then remove the zero rows we obtain the matrix in Figure 7.8. It can be seen that the rows of block B^*, associated with vectors $\bar{t}_1^1(a) - \bar{t}_1^1$, have the three entries corresponding to each $a \in E_R$ and its corresponding parallel edge a' as follows:

$$\begin{array}{c}
\begin{array}{ccccc}
\text{Path 1} & \text{Path 2} & & \text{Path } P & \mathbf{z}
\end{array}\\
\left(\begin{array}{c:c:c:c:c}
\bar{T}^1 - \bar{t}_1^1 & 0 & \cdots & 0 & 0 \\ \hdashline
0 & T^2 - t_1^2 & \cdots & 0 & 0 \\ \hdashline
\cdots & \cdots & \cdots & \cdots & \cdots \\ \hdashline
0 & 0 & \cdots & T^P - t_1^P & 0 \\ \hline
B^* & 0 & \cdots & 0 & 0 \\ \hdashline
0 & \begin{matrix} -1\,-1 & & \\ & \ddots & \\ & & -1\,-1 \end{matrix} & \cdots & 0 & 0 \\ \hdashline
\cdots & \cdots & \cdots & \cdots & \cdots \\ \hdashline
0 & 0 & \cdots & \begin{matrix} -1\,-1 & & \\ & \ddots & \\ & & -1\,-1 \end{matrix} & 0 \\ \hline
* & * & \cdots & * & \begin{matrix} \mathbf{z}^2 - \mathbf{z}^1 \\ \cdots \\ \mathbf{z}^{w+1} - \mathbf{z}^1 \end{matrix}
\end{array}\right)
\end{array}$$

Figure 7.8: Matrix appearing in the proof of Theorem 7.3.13

- $x(a) = x(a') = -1$, $y(a') = 0$, for each $a \in E_R(V \setminus S)$,
- $x(a) = x(a') = 1$, $y(a') = 0$, for each $a \in \delta_R(S)$, and
- $x(a) = 1$, $x(a') = 0$, $y(a') = -1$, for each $a \in E_R(S)$ in the case $f \in E_{NR}$, while,
- in the case $f \in E_R$, $x(a) = 1$, $x(a') = 0$, $y(a') = -1$, for each $a \in E_R(S) \setminus \{f\}$, and $x(f) = -1$, $x(f') = -1$, $y(f') = 0$.

In the case $f \in E_{NR}$, the matrix obtained after merging blocks (1,1) and B^* of Figure 7.8 is detailed in Figure 7.9, with the columns and rows rearranged. Note that, for $a \in E_R$ the corresponding values -1 or 1 are the only nonzero entries in its corresponding column of the matrix in Figure 7.8. In the case $f \in E_R$, the entry corresponding to edge f in the block (4,4) of this matrix, is -1 instead of 1. In both cases, this matrix has full rank. Hence, the matrix of Figure 7.8 has full rank and its $Pm-1+P|E_R|+w = P(2|E_{NR}|+|E_R|)+(|V_R|^2-1)^2-1$ rows are linearly independent. Hence, we have $(|V_R|+1)(2|E_{NR}|+|E_R|)+(|V_R|^2-1)^2$ LDdGRP solutions affinely independent satisfying inequality (7.4) as an equality and we are done.

If $P = 2$, the solutions corresponding to some rows of the last block row of the corresponding matrix, $[\bar{t}_1^1, t_1^2(a_i)]$, are not actually LDdGRP solutions. Note that, if $a_i \in \delta_R(S) \cup E_R(S)$, neither path 1, performing $\bar{t}_1^1$, nor path 2, performing $t_1^2(a_i)$, traverse this edge. In this case, for each $a_i \in \delta_R(S) \cup E_R(S)$ we consider the solution in which path 1 performs a vector similar to $\bar{t}_1^1$ but traversing a_i, and path 2 performs

$$\begin{array}{c}
\begin{array}{cccc}
\quad E_{NR} \quad & \quad E_R(V\backslash S) \quad & \quad \delta_R(S) \quad & \quad E_R(S) \quad
\end{array} \\
\left(\begin{array}{c:c:c:c}
\begin{matrix}(\bar{t}_2^1-\bar{t}_1^1)_{NR}\\ \cdots \\ (\bar{t}_m^1-\bar{t}_1^1)_{NR}\end{matrix} & 0 & 0 & 0 \\
\hdashline
* & \begin{matrix}-1 & & \\ & \ddots & \\ & & -1\end{matrix} & 0 & 0 \\
\hdashline
* & 0 & \begin{matrix}1 & & \\ & \ddots & \\ & & 1\end{matrix} & 0 \\
\hdashline
* & 0 & 0 & \begin{matrix}1 & & \\ & \ddots & \\ & & 1\end{matrix}
\end{array}\right)
\end{array}$$

Figure 7.9: Submatrix B^* of the matrix in Figure 7.8

$t_1^2(a_i)$, which is a LDdGRP solution. It can be seen that the corresponding matrix is also full rank. □

Note 7.3.13.1. In the case $|(V\backslash S)\cap V_R^0| = 1$, we do not know if connectivity inequalities (7.4) induce a facet of LDdGRP(G) or not.

In what follows, several families of valid inequalities for the LDdGRP are presented, namely parity, aggregate parity and q–connectivity inequalities.

7.3.2 Parity inequalities

Given a set $S \subset V$, any closed walk traverses an even (maybe zero) number of times the cut–set $\delta(S)$. In the formulation we have proposed for the LDdGRP, each solution is formed by a set of “open” paths. If a path starts and ends in the same shore of the cut–set, it has to traverse the cut–set an even number of times (maybe zero). However, if a path starts at one shore of the cut–set and ends at the other one, it necessarily has to traverse the cut–set an odd number of times. This is the idea behind the following parity inequalities.

Let us consider a set $S \subset V$ of vertices and an edge subset $F \subseteq \delta(S)$. For each path p, the inequalities

$$x^p(\delta_R(S)\setminus F) + (x^p - y^p)(\delta_{NR}(S)\setminus F) \geq x^p(F_R) + (x^p - y^p)(F_{NR}) - |F| +$$
$$+ z^{p-1}(S\cap V_R^0) + z^p(S\cap V_R^0) - 1, \tag{7.29}$$

if $|F|$ is odd, and

$$x^p(\delta_R(S)\setminus F) + (x^p - y^p)(\delta_{NR}(S)\setminus F) \geq x^p(F_R) + (x^p - y^p)(F_{NR}) - |F| +$$

$$+z^{p-1}(S \cap V_R^0) - z^p(S \cap V_R^0), \tag{7.30}$$

if $|F|$ is even (maybe zero), are called *parity inequalities* for the LDdGRP.

Inequalities (7.29) cut infeasible solutions in which a path p starts and ends at S, and traverses the cut–set $\delta(S)$ an odd number of times. The set F is defined by the set of edges in $\delta(S)$ traversed exactly once, and then, $x^p(F_R) + (x^p - y^p)(F_{NR}) - |F| = 0$ holds. Moreover, $x^p(\delta_R(S) \setminus F) + (x^p - y^p)(\delta_{NR}(S) \setminus F)$ is also equal to zero, while $z^{p-1}(S \cap V_R^0) + z^p(S \cap V_R^0) - 1 = 1$. Hence, the solution does not satisfy the parity inequality (7.29).

Inequalities (7.30) cut infeasible solutions in which a path p starts at S, ends at $V \backslash S$, and traverses the cut–set $\delta(S)$ an even number of times. The set F is defined again by the set of edges in $\delta(S)$ traversed exactly once. Note that F can now be empty due to all the edges $e \in \delta(S)$ being traversed twice (or non traversed). Anyway, the left–hand side (LHS) of the inequality is 0, while the right–hand side (RHS) is 1, and the solution does not satisfy the parity inequality (7.30).

Theorem 7.3.14. *Parity inequalities* (7.29) *and* (7.30) *are valid for the* LDdGRP*.*

Proof: Let $S \subset V$ be a set of vertices and let $F \subseteq \delta(S)$ be an edge subset. We have to prove that, for any LDdGRP solution, the path p satisfies the corresponding parity inequality.

Let us first consider $|F|$ odd. The LDdGRP solutions $(\bar{x}, \bar{y}, \bar{z})$ satisfying either $\bar{x}^p(F_R) + (\bar{x}^p - \bar{y}^p)(F_{NR}) \leq |F| - 1$ or $\bar{z}^{p-1}(S \cap V_R^0) + \bar{z}^p(S \cap V_R^0) \leq 1$, trivially satisfy inequality (7.29). The remaining LDdGRP solutions $(\bar{x}, \bar{y}, \bar{z})$ satisfy $\bar{x}^p(F_R) + (\bar{x}^p - \bar{y}^p)(F_{NR}) = |F|$ (path p traverses each edge in F once) and $\bar{z}^{p-1}(S \cap V_R^0) + \bar{z}^p(S \cap V_R^0) = 2$ (path p starts and ends in S), and the corresponding RHS of (7.29) is equal to 1. Given that $|F|$ is odd, the path p of $(\bar{x}, \bar{y}, \bar{z})$ must traverse the cut–set at least once more and $\bar{x}^p(\delta_R(S) \setminus F) + (\bar{x}^p - \bar{y}^p)(\delta_{NR}(S) \backslash F) \geq 1$ holds. Therefore, the solution satisfies inequality (7.29).

Let us consider now $|F|$ even. The LDdGRP solutions $(\bar{x}, \bar{y}, \bar{z})$ satisfying either $\bar{x}^p(F_R) + (\bar{x}^p - \bar{y}^p)(F_{NR}) \leq |F| - 1$ or $z^{p-1}(S \cap V_R^0) - z^p(S \cap V_R^0) \leq 0$, trivially satisfy inequality (7.30). The remaining LDdGRP solutions $(\bar{x}, \bar{y}, \bar{z})$ satisfy $\bar{x}^p(F_R) + (\bar{x}^p - \bar{y}^p)(F_{NR}) = |F|$ (path p traverses each edge in F once) and $z^{p-1}(S \cap V_R^0) - z^p(S \cap V_R^0) = 1$ (path p starts at S and ends at $V \setminus S$), and the RHS of (7.30) is equal to 1. Given that $|F|$ is even, the path p of $(\bar{x}, \bar{y}, \bar{z})$ traverses the cut–set at least once again and $\bar{x}^p(\delta_R(S) \setminus F) + (\bar{x}^p - \bar{y}^p)(\delta_{NR}(S) \setminus F) \geq 1$ holds. Therefore, the solution satisfies inequality (7.30). $\square$

Ω–Aggregate parity inequalities

The parity inequalities (7.29) and (7.30) considered above involve a single path $p \in \mathcal{P}$. In this section we present a generalization of these inequalities to a subset of paths $\Omega \subseteq \mathcal{P}$, in which the variables x and y that appear in inequalities (7.29) or (7.30) are added for all paths in Ω (see (7.31)). These new "aggregate" inequalities would separate fractional solutions in which a subset of paths do not jointly satisfy the parity conditions in a given cut–set $\delta(S)$.

Aggregate inequalities for two or more paths are not satisfied by all LDdGRP solutions we have considered so far. Note that the validity of inequalities (7.29) and (7.30) is based on the fact that all the LDdGRP solutions $(\bar{x}, \bar{y}, \bar{z})$ satisfy

$$\bar{x}^p(F_R) + (\bar{x}^p - \bar{y}^p)(F_{NR}) \leq |F|$$

and, hence, the RHS of (7.29) and (7.30) is at most 1. However, if we add the variables x and y for all the paths in a set Ω, with $|\Omega| \geq 2$, the condition

$$\sum_{p\in\Omega} \bar{x}^p(F_R) + \sum_{p\in\Omega}(\bar{x}^p - \bar{y}^p)(F_{NR}) \leq |F|$$

does not hold because $\sum_{p\in\Omega} \bar{x}^p_e$ can take a value up to $|\Omega|$. In order for the "aggregate" parity inequality to be satisfied, $\sum_{p\in\Omega} \bar{x}^p_e \leq 1$ is needed for all the edges in F. Then, we will assume that

1) the set F does not contain non–required edges, and
2) for the (required) edges in F, $\sum_{p\in\Omega} \bar{x}^p_e \leq 1$ holds.

Thus, in this section we will assume that all the LDdGRP solutions satisfy $\sum_{p\in\mathcal{P}} \bar{x}^p_e = 1$, for all $e \in E_R$, instead of $\sum_{p\in\mathcal{P}} \bar{x}^p_e \leq 1$. Note that there is always an *optimal* LDdGRP solution in which each required edge is traversed in exactly one path (because each required edge e has a parallel non–required edge e', with $c^s_e \geq c_{e'}$, and one path can traverse e and the remaining paths traverse e').

Let us consider a set $S \subset V$ of vertices, an edge subset $F \subseteq \delta_R(S)$, and a non–empty subset Ω of paths. Let us consider a partition of Ω into four (maybe empty) subsets, $\Omega = \Omega_1 \cup \Omega_2 \cup \Omega_3 \cup \Omega_4$, in such a way that $|F| + |\Omega_1 \cup \Omega_2|$ is an odd number. If we denote $S_R = S \cap V^0_R$ and $\bar{S} = V \setminus S$, we will call Ω*–aggregate parity inequality* to

$$\begin{aligned}\sum_{p\in\Omega} x^p(\delta_R(S)\setminus F) + \sum_{p\in\Omega}(x^p - y^p)(\delta_{NR}(S)) \geq \sum_{p\in\Omega} x^p(F) - |F| + \\ + \sum_{p\in\Omega_1}\left(z^{p-1}(S_R) - z^p(S_R)\right) + \sum_{p\in\Omega_2}\left(z^{p-1}(\bar{S}_R) - z^p(\bar{S}_R)\right) + \\ + \sum_{p\in\Omega_3}\left(z^{p-1}(S_R) - z^p(\bar{S}_R)\right) + \sum_{p\in\Omega_4}\left(z^{p-1}(\bar{S}_R) - z^p(S_R)\right) - |\Omega| + 1 \qquad (7.31)\end{aligned}$$

The idea behind the definition of the Ω_i subsets is to group the paths depending on their starting and ending shores. Specifically,

- Paths in Ω_1 are considered as starting at S and ending at $\bar{S}$ paths,
- Ω_2 is formed by paths starting at $\bar{S}$ and ending at S,
- Ω_3 is formed by paths starting and ending at S, and
- Ω_4 is formed by paths starting and ending at $\bar{S}$.

Theorem 7.3.15. *$\Omega-$aggregate parity inequalities* (7.31) *are satisfied by all the LDdGRP solutions such that $\sum_{p\in\mathcal{P}} x^p_e = 1$ for all $e \in E_R$.*

Proof: Let $(\bar{x}, \bar{y}, \bar{z})$ be an LDdGRP solution such that $\sum_{p\in\mathcal{P}} \bar{x}^p_e = 1$ for all $e \in E_R$. Since $F \subseteq E_R$, $\sum_{p\in\Omega} \bar{x}^p(F) \leq |F|$. Furthermore, each term $\sum_{p\in\Omega_i} \left(\bar{z}^{p-1}(\cdot) - \bar{z}^p(\cdot)\right)$ is less than or equal to $|\Omega_i|$ and, hence, the RHS of inequality (7.31) is, at most, $0 + |\Omega| - |\Omega| + 1 = 1$.

The LDdGRP solutions $(\bar{x}, \bar{y}, \bar{z})$ for which the RHS of (7.31) is less than or equal to zero trivially satisfy the inequality. The RHS of (7.31) is 1 only for the LDdGRP solutions $(\bar{x}, \bar{y}, \bar{z})$ that satisfy:

- $\sum_{p\in\Omega} \bar{x}^p(F) = |F|$,
- $\bar{z}^{p-1}(S_R) = 1$ and $\bar{z}^p(S_R) = 0$, for all $p \in \Omega_1$,
- $\bar{z}^{p-1}(\bar{S}_R) = 1$ and $\bar{z}^p(\bar{S}_R) = 0$, for all $p \in \Omega_2$,
- $\bar{z}^{p-1}(S_R) = 1$ and $\bar{z}^p(\bar{S}_R) = 0$, for all $p \in \Omega_3$,
- $\bar{z}^{p-1}(\bar{S}_R) = 1$ and $\bar{z}^p(S_R) = 0$, for all $p \in \Omega_4$.

These solutions traverse each edge $e \in F$ once, the paths in Ω_1 start in S and end in $\bar{S}$, the paths in Ω_2 start in $\bar{S}$ and end in S, and the paths in Ω_3 and Ω_4 start and end in S and $\bar{S}$, respectively. Given that $|F| + |\Omega_1 \cup \Omega_2|$ is odd, these solutions have to traverse the cut–set $\delta(S)$ at least once more, and $\sum_{p\in\Omega} \bar{x}^p(\delta_R(S) \setminus F) + \sum_{p\in\Omega} (\bar{x}^p - \bar{y}^p)(\delta_{NR}(S)) \geq 1$ holds. □

The Ω–aggregate parity inequalities (7.31) consider a subset of paths jointly. If this subset contains only one path, we obtain the parity inequalities (7.29) and (7.30), but with $F \subseteq E_R$. If $|F|$ is odd, then $|\Omega_1 \cup \Omega_2|$ has to be even (or zero). Let $\Omega_1 = \Omega_2 = \Omega_4 = \emptyset$ and $\Omega_3 = \{p\}$. The corresponding inequality (7.31) is

$$x^p(\delta_R(S) \setminus F) + (x^p - y^p)(\delta_{NR}(S)) \geq x^p(F_R) - |F| + z^{p-1}(S_R) - z^p(\bar{S}_R),$$

which is inequality (7.29) with $F \subseteq E_R$ and $|F|$ odd, and with $-z^p(\bar{S}_R) = z^p(S_R) - 1$ (since $z^p(S_R) + z^p(\bar{S}_R) = 1$ holds for each path p).

On the other hand, if $|F|$ is even, then $|\Omega_1 \cup \Omega_2|$ has to be odd, and by considering $\Omega_1 = \{p\}$ and $\Omega_2 = \Omega_3 = \Omega_4 = \emptyset$, the corresponding inequality (7.31) is

$$x^p(\delta_R(S) \setminus F) + (x^p - y^p)(\delta_{NR}(S)) \geq x^p(F_R) - |F| + z^{p-1}(S_R) - z^p(S_R),$$

which is inequality (7.30) with $F \subseteq E_R$ and $|F|$ even.

On the other hand, in the case that set Ω includes all the paths, $\Omega = \mathcal{P}$, if we consider $F = \delta_R(S)$ and $|F|$ is odd, we have the following "*aggregate*" *parity inequality*:

$$\sum_{p\in\mathcal{P}} (x^p - y^p)(\delta_{NR}(S)) \geq 1, \tag{7.32}$$

which is obviously satisfied by all the LDdGRP solutions such that $\sum_{p\in\mathcal{P}} x^p_e = 1$ for all $e \in E_R$.

7.3.3 q–connectivity inequalities

p–connectivity inequalities were presented in Sections 4.2.1 and 4.2.2 for the K–RPP, based on those with the same name proposed for the MBCPP in Corberán et al. (2013). Here we present a version of such inequalities for the LDdGRP. We will use here the letter q instead of p to avoid confusion with the path indices.

Let $\{S_0, \dots, S_q\}$ be a partition of V. The following inequality

$$\sum_{p\in\mathcal{P}} x^p(\delta_R(S_0)) + \sum_{p\in\mathcal{P}}(x^p + y^p)(\delta_{NR}(S_0)) + 2\sum_{p\in\mathcal{P}}\sum_{1\le r<t\le q} x^p(S_r : S_t) \ge 2q \qquad (7.33)$$

will be referred to as *aggregate q–connectivity inequality* for the LDdGRP.

Theorem 7.3.16. *Aggregate q–connectivity inequalities* (7.33) *are valid for the* LDdGRP.

Proof: Let $(\bar{x}, \bar{y}, \bar{z})$ be an LDdGRP tour. If $\sum_{p\in\mathcal{P}} \bar{x}^p(S_r : S_t) \ge 1$ for certain sets S_r, S_t, with $1 \le r < t \le q$, we can merge S_r and S_t, obtaining a new q–connectivity configuration with $q-1$ sets. It can be seen that if its associated $(q-1)$–connectivity inequality is satisfied by $(\bar{x}, \bar{y}, \bar{z})$, also is the original one. Therefore, we can assume that $\sum_{p\in\mathcal{P}} \bar{x}^p(S_r : S_t) = 0$ for any pair of sets S_r, S_t, with $r, t \ne 0$. Note that, in this case, each set S_i contains either a vertex in V_R^0, a required edge, or a vertex in V_I incident with a required edge in $(S_0 : S_i)$. Thus, the tour $(\bar{x}, \bar{y}, \bar{z})$ has to visit each set S_i and, then, $\sum_{p\in\mathcal{P}}(\bar{x}^p+\bar{y}^p)(S_0 : S_i) \ge 2$ holds and $\sum_{p\in\mathcal{P}} \bar{x}^p(\delta_R(S_0))+\sum_{p\in\mathcal{P}}(\bar{x}^p+\bar{y}^p)(\delta_{NR}(S_0)) \ge 2q$. □

The above inequalities consider the "whole" LDdGRP tour, the sum of the P paths. However, they can also be defined for each single path p. Let $\{S_0, \dots, S_q\}$ be a partition of V and let us consider an edge $e_j \in E(S_j)$ for each $j = 1, \dots, q$. Assume that there are al least two required vertices in S_0, $|S_0 \cap V_R^0| \ge 2$. For each $p \in \mathcal{P}$, the inequality

$$x^p(\delta_R(S_0)) + (x^p + y^p)(\delta_{NR}(S_0)) + 2\sum_{1\le r<t\le q} x^p(S_r : S_t) \ge 2\sum_{j=1}^{q} x^p_{e_j}$$

$$z^{p-1}(V_R^0 \cap S_0) + z^p(V_R^0 \cap S_0) - 2 \qquad (7.34)$$

will be referred to as *q–connectivity inequality* for the LDdGRP.

Theorem 7.3.17. *q–connectivity inequalities* (7.34) *are valid for the* LDdGRP.

Proof: We have to prove that, for any LDdGRP solution $(\bar{x}, \bar{y}, \bar{z})$, the path p satisfies the inequality.

For the LDdGRP solutions $(\bar{x}, \bar{y}, \bar{z})$ satisfying $\bar{x}^p_{e_j} = 1$ for all $j = 1, \dots, q$, we distinguish three cases depending on the values of $\bar{z}^{p-1}(S_0 \cap V_R^0)$ and $\bar{z}^p(S_0 \cap V_R^0)$:

a) If $\bar{z}^{p-1}(S_0 \cap V_R^0) = \bar{z}^p(S_0 \cap V_R^0) = 1$, the RHS of (7.34) for these solutions is $2q$, and the path p starts and ends at S_0 and visits all the remaining sets S_i. As can be seen in the proof of Theorem 7.3.16, the LHS of (7.34) for these solutions is at least $2q$ and, hence, they satisfy (7.34).

b) If $\bar{z}^{p-1}(S_0 \cap V_R^0) = \bar{z}^p(S_0 \cap V_R^0) = 0$, the RHS of (7.34) for these solutions is $2q-2$, and the path p starts and ends at vertices in $V \setminus S_0$, and visits all the sets S_j, $j > 0$. The LHS of (7.34) for these solutions is at least $2q-2$ because, otherwise, after adding two edges in any set $(S_0 : S_i)$, with coefficient 1 in the inequality, we would have a closed walk on the sets $S_0, \dots, S_q$ for which the LHS of a q–connectivity inequality would be less than $2q$, and this would contradict a).

c) If $\bar{z}^{p-1}(S_0 \cap V_R^0) = 1$ and $\bar{z}^p(S_0 \cap V_R^0) = 0$ (or viceversa), the RHS of (7.34) for these solutions is $2q-1$, and the path p starts at S_0, visits all the sets S_j, $j > 0$, and ends at a given S_i, $i > 0$. The LHS of (7.34) for these solutions is at least $2q-1$ because, otherwise, after adding an edge in the set $(S_0 : S_i)$, with coefficient 1 in the inequality, we would have a closed walk on the sets $S_0, \dots, S_q$ for which the LHS of a q–connectivity inequality would be less than $2q$.

For the LDdGRP solutions $(\bar{x}, \bar{y}, \bar{z})$ satisfying $\bar{x}_{e_j}^p = 1$ for all $j = 1, \dots, q$ except one of them, we distinguish the same three cases as above. If both $\bar{z}^{p-1}(S_0 \cap V_R^0)$ and $\bar{z}^p(S_0 \cap V_R^0)$ are equal to 1, the RHS of (7.34) for these solutions is $2q-2$ and the path p has to visit all the sets S_j except one of them, so the LHS of (7.34) for these solutions is at least $2q-2$. The same happens for the cases b) and c), where the RHS of (7.34) for these solutions is two units less than the RHS in b) and c) above, while the LHS also decreases by two units. For the remaining LDdGRP solutions $(\bar{x}, \bar{y}, \bar{z})$ satisfying $x_{e_j}^p = 1$ for all $j = 1, \dots, q$ except two or more of them, a similar reasoning proves that they also satisfy (7.34). □

7.4 A branch–and–cut algorithm for the LDdGRP

We present in this section the branch–and–cut algorithm we have implemented using the families of inequalities described in the polyhedral study of the previous section. For the cutting–plane algorithm embeded in the branch–and–cut method, we have considered separation procedures for connectivity inequalities (7.4) and (7.17), parity inequalities (7.29) and (7.30), aggregate parity inequalities (7.32), aggregate q–connectivity inequalities (7.33), and q–connectivity inequalities (7.34).

The initial LP relaxation includes inequalities (7.5)–(7.11) from the formulation, equations (7.16), inequalities (7.19)–(7.21) needed to linearize the objective function, and the bounds on the variables. Furthermore, we removed from the formulation the variables from Theorem 7.2.1.

At each node of the search tree, the cutting–plane procedure is applied, the violated cuts found are added to the relaxation, and the LP is solved again until no more violated cuts are found by the cutting–plane. Then we branch using the strong branching strategy implemented in Cplex but giving higher priority to the variables z_i^p.

7.4.1 Separation algorithms

We describe here the separation algorithms that have been used to identify violated inequalities at each iteration of the cutting–plane algorithm.

Connectivity inequalities

For each path p, we compute the connected components of the graph induced by those edges $e \in E$ such that $\bar{x}^p_e + \bar{y}^p_e > \varepsilon$, where ε is a given parameter. For each one of these connected components, say S, we find the edge $f \in E(S)$ and the vertex $v \in S$ such that $\bar{x}^p_f$ and $\bar{z}^p_v$ are maximal, respectively. Now, if $\bar{x}^p_f > \bar{z}^p_v$, we check the corresponding connectivity inequality (7.4) for violation. Otherwise, we check connectivity inequality (7.17).

Parity inequalities

Given a path p and a subset of vertices $S \subset V$, we have adapted the procedure described in Ghiani and Laporte (2000) for the so–called co–circuit inequalities to obtain the set $F \subseteq \delta(S)$ that corresponds to the most violated parity inequality (7.2) and (7.15) associated with this subset. We create set F as the set containing those edges $e \in \delta(S)$ such that $\bar{x}^p_e - \bar{y}^p_e \geq 0.5$.

If $|F|$ is odd, we check parity inequality (7.2) for violation. Moreover, let

$$e_1 = \arg\min_{e \in F} \{\bar{x}^p_e - \bar{y}^p_e\}, \quad e_2 = \arg\max_{e \in \delta(S) \setminus F} \{\bar{x}^p_e - \bar{y}^p_e\}.$$

If $\bar{x}^p_{e_1} - \bar{y}^p_{e_1} - 0.5 \geq 0.5 - (\bar{x}^p_{e_2} - \bar{y}^p_{e_2})$, we also try adding edge e_1 to F and checking inequality (7.15) for violation. Otherwise, we try removing edge e_2 from F.

In the case that the obtained set F is even, we check parity inequality (7.15) for violation and try adding to or removing one edge from F as before in order to look for violated inequalities (7.2).

For each path p, we look for violated parity inequalities (7.2) and (7.15) by applying the above procedure for all the subsets formed by one single vertex in V_I (incident with at least one required edge) and for the subsets $S \subset V$ generated as follows. For each edge $e \in E$, we define the weight

$$w_e = \min\{1 - (\bar{x}^p_e - \bar{y}^p_e),\ \bar{x}^p_e - \bar{y}^p_e\}$$

and compute the connected components induced by those edges with $w_e > \varepsilon$, where ε is a given parameter. Each connected component defines a subset S that will be studied.

We also look for violated aggregated parity inequalities (7.32). For each edge $e \in E$, we define the weight

$$w^{agg}_e = \min\left\{1 - \sum_{p \in \mathcal{P}} (\bar{x}^p_e - \bar{y}^p_e),\ \sum_{p \in \mathcal{P}} (\bar{x}^p_e - \bar{y}^p_e)\right\}$$

and obtain the connected components induced by the edges with $w_e^{agg} > \varepsilon$. Then, the same procedure as above is used to find the set $F \subseteq \delta(S)$ with the difference that now we only look for sets F with $|F|$ odd.

q–connectivity inequalities

We have implemented heuristic separation algorithms for both q–connectivity (7.34) and aggregate q–connectivity (7.33) inequalities.

Given an LP solution $(\bar{x}, \bar{y}, \bar{z})$ and a path p, we calculate the connected components $C_1, \ldots, C_t$ of the graph induced by the edges with $\bar{x}_e^p + \bar{y}_e^p > 0$. Now we shrink any pair of components C_i, C_j such that $(\bar{x}^p + \bar{y}^p)(C_i : C_j) \geq 2$. Let S_0 be the union of the components C_i such that $\bar{z}^p(C_i) \geq \varepsilon_z$ or $\bar{z}^{p-1}(C_i) \geq \varepsilon_z$ and $S_1, \ldots, S_q$ the remaining components C_j, where $\varepsilon_z \geq 0$ is a parameter.

Now we merge every set S_i, $i \neq 0$, containing no edges, i.e. $E(S_i) = \emptyset$, with another one in the following way. We look for the set S_r, $r \notin \{0, i\}$, that maximizes $\bar{x}^p(S_i : S_r)$. If $(\bar{x}^p + \bar{y}^p)(S_0 : S_i) + \bar{z}^p(S_i) + \bar{z}^{p-1}(S_i) > (\bar{x}^p + \bar{y}^p)(S_i : S_r)$, we do $S_0 = S_0 \cup S_i$. Otherwise, we merge S_i and S_r.

For each set S_i, $i \neq 0$, we choose $e_i = \arg\max_{e \in E(S_i)}\{\bar{x}_e^p\}$.

Now, while $q > 2$, we try to continue merging sets as follows:

- If $2\bar{x}^p(S_i : S_j) > \min\{\bar{x}_{e_i}^p, \bar{x}_{e_j}^p\}$, with $i, j \neq 0$, we merge S_i and S_j and choose as representative edge of the resulting merged set the edge among e_i and e_j with maximum value of $\bar{x}_e^p$.
- If $\sum_{r \neq i,0}\big((\bar{x}^p - \bar{y}^p)(S_i : S_r)\big) + \bar{x}^p(S_i : S_0) + \bar{z}^{p-1}(S_i) + \bar{z}^p(S_i) > 2\bar{x}_{e_i}^p$, $i \neq 0$, we do $S_0 = S_0 \cup S_i$.

Finally, we check the q–connectivity inequality (7.34) corresponding to the resulting sets $S_0, S_1, \ldots, S_q$ and their associated edges $e_1, \ldots, e_q$ for violation.

In order to find violated aggregated q–connectivity inequalities (7.33), we apply the following procedure. Given an LP solution $(\bar{x}, \bar{y}, \bar{z})$, let us call $\bar{x}_e^{agg} = \sum_{p \in \mathcal{P}} \bar{x}_e^p$ and $\bar{y}_e^{agg} = \sum_{p \in \mathcal{P}} \bar{y}_e^p$, i.e. the result of adding all the paths of the solution. First we calculate the connected components $C_1, C_2, \ldots, C_t$ induced by the required edges and the required nodes of G. Now we iteratively merge all the pairs of components C_i,C_j such that $(\bar{x}^{agg} + \bar{y}^{agg})(C_i : C_j) \geq 2$.

For each resulting set C_i, we proceed as follows. We choose $S_0 = C_i$, while $S_1, \ldots S_q$ will be the remaining sets C_j. While $q > 2$, we will try to merge pairs of sets S_i, S_j according to the following rules:

- If $i, j \neq 0$ and $\bar{x}^{agg}(S_i : S_j) \geq 1$, we merge S_i and S_j.
- If $i = 0$ (or $j = 0$) and $\sum_{r \neq i,0}\big((\bar{x}^{agg} - \bar{y}^{agg})(S_i : S_r)\big) + (\bar{x}^{agg} + \bar{y}^{agg})(S_0 : S_i) > 2$ we do $S_0 = S_0 \cup S_i$.

We check the aggregated q–connectivity inequality (7.33) corresponding to the resulting sets $S_0, S_1, \ldots, S_q$ for violation.

Instance name	$\lvert E_R \rvert$	$\lvert V \rvert$
LDdGRP441	10.0	$[12, 17]$
LDdGRP442	11.2	$[14, 21]$
LDdGRP551	17.0	$[19, 24]$
LDdGRP552	21.3	$[24, 31]$
LDdGRP661	21.0	$[28, 33]$
LDdGRP662	25.2	$[31, 38]$
LDdGRP771	30.0	$[40, 45]$
LDdGRP772	43.5	$[45, 53]$
LDdGRP881	46.0	$[57, 62]$
LDdGRP882	63.0	$[58, 63]$
LDdGRP991	56.0	$[66, 71]$
LDdGRP992	79.0	$[71, 82]$
LDdGRP10101	67.0	$[82, 87]$
LDdGRP10102	95.5	$[86, 95]$

Table 7.4: Characteristics of the LDdGRP instances

7.5 Instances

In order to evaluate the behaviour of the proposed branch–and–cut algorithm, we have randomly generated a set of LDdGRP instances as follows. Initially, we generate a GRP instance on a grid with $n \times n$ points whose coordinates are multiples of 100. All the vertical and horizontal edges, as well as some diagonals, are contained in the original graph. Then, each edge of such a graph is considered required (and therefore included in the instance) with probability p. Moreover, we randomly select the depot among the vertices of the grid, and the coordinates of all the vertices are slightly perturbed to avoid completely horizontal or vertical edges. Those vertices that are not incident with required edges are removed from the graph. A set of nv_{req} required vertices with random coordinates is generated and added to the instance. Deadheading times per unit of weight between any pair of vertices of the graph are considered equal to the Euclidean distances between these two points in the grid, while the servicing time (per unit of weight) of a required edge is equal to the deadheading time multiplied by 1.5.

A total of 7 different grid sizes, ranging from 4×4 to 10×10, have been considered. For each grid size, 12 instances have been generated, half of them classified as version 1 (with $p = 0.3$) and the other half as version 2 (with $p = 0.4$). For each version on the same grid, we generate one instance for each value of nv_{req} in $\{2, 3, 4, 5, 6, 7\}$. Thus, we have a total of 84 instances, whose characteristics are summarized in Table 7.4. Each row of the table corresponds to a set of six different instances, each one with a different number of required vertices. The digits at the end of the name on the first column indicate the grid size and whether the instances have been generated as version 1 or 2. Column 2 reports the average number of required edges, while column 3 shows the minimum and maximum total number of vertices among the considered instances in the row.

The demand d_i of each required vertex i is chosen randomly in $\{1, 2\}$. The total demand of an instance will then be between 2 and 14. Since for many delivery drones the ratio of its curb weight and its maximum payload ranges between 1 : 1 and 1 : 5, we have considered three possible values for the curb weight w_0 of the drone: 15, 30 and 45. Each one of these values has been randomly assigned to 28 of the instances. Given that the average value of c_e^s for the required edges of the graphs is aproximately 150, we have considered the times (per unit of weight) c_D of servicing (landing and taking off) a required vertex in $\{100, 125, 150\}$. Again, we assign each value to one third of the instances in a random way.

The characteristics of the 84 LDdGRP instances generated, as well as the results obtained in the experiments explained in what follows, can be found in http://www.uv.es/plani/instancias.htm.

7.6 Computational experiments

In this Section we first present the experiments carried out on a reduced set of instances in order to assess the contribution of each family of valid inequalities. The results obtained are used to decide the configuration of the cutting-plane procedure we will use in our final implementation of the branch–and–cut algorithm. This final algorithm is then tested on the complete set of instances and the results discussed.

All the algorithms have been coded in C++ and the tests have been run on an Intel Core i7 at 3.4 GHz with 16 GB RAM. The B&C uses CPLEX 20.1 MIP Solver with a single thread. CPLEX heuristic algorithms were turned off, as well as CPLEX's own cuts. The optimality gap tolerance was set to zero and best bound strategy was selected.

7.6.1 Configuration of the cutting–plane algorithm

In order to get an idea of the impact of using each family of inequalities, we have designed the following experiments. A subset of 15 instances of the collection described in Section 7.5 has been randomly selected. We have first implemented a barebones version of the branch–and–cut algorithm in which only the connectivity separation algorithm and the separation procedure for parity inequalities defined on sets of one single vertex are used to find violated inequalities when the solution of the LP relaxation is integer (they are added as lazy constraints in Cplex). This guarantees that any integer solution for which no violated cuts are found will be a feasible LDdGRP solution. We call this version of the branch and cut "Base" version.

Now we run the reduced set of instances several times, using each time the Base version plus only one of the separation algorithms listed here (applied to the fractional solutions of the LP relaxation, i.e. as user cuts in Cplex):

(1) Connectivity inequalities (7.4) and (7.17)

Cutting–plane version		①	②	③	④	⑤	⑥
Base	7	7	7	**9**	8	6	7
Base + ③	9	9	9	–	**11**	9	9
Base + ③ + ④	11	**13**	12	–	–	11	12
Base + ① + ③ + ④	13	–	14	–	–	12	$\boxed{\textbf{14}}$
Base + ① + ③ + ④ + ⑥	**14**	–	14	–	–	12	–
Base + all inequalities	12	–	–	–	–	–	–

Table 7.5: Number of optimal solutions found

Cutting–plane version		①	②	③	④	⑤	⑥
Base	9.70	3.83	10.07	**2.16**	13.78	29.03	7.05
Base + ③	2.16	1.48	2.52	–	**0.88**	3.41	2.19
Base + ③ + ④	0.88	**0.18**	0.41	–	–	0.95	0.36
Base + ① + ③ + ④	0.18	–	0.18	–	–	0.71	$\boxed{\textbf{0.05}}$
Base + ① + ③ + ④ + ⑥	**0.05**	–	0.06	–	–	0.23	–
Base + all inequalities	0.31	–	–	–	–	–	–

Table 7.6: Average percentage gap

② Parity inequalities (7.15) for sets of one single vertex

③ Parity inequalities (7.29) and (7.30)

④ aggregate parity inequalities (7.32)

⑤ q–connectivity inequalities (7.34)

⑥ aggregate q–connectivity inequalities (7.33)

Once all the six versions (plus the base version) have been run, we select the one that provides the best results. For the next batch of runs, we use the version that provides the best results and try adding each one of the remaining separation algorithms to it. We continue selecting the best version of each batch and adding the separation algorithms that contribute the most to the performance of the branch–and–cut procedure until no significant improvement is observed.

The results obtained in these experiments are reported in Tables 7.5 to 7.8. Each one of these tables presents the performance of the procedure in what refers to one specific measure: number of instances solved to optimality (Table 7.5), average percentage gap between the final upper and lower bounds (Table 7.6), average computing time in seconds (Table 7.7), and average value of the final lower bound (Table 7.8). The first

Cutting–plane version		①	②	③	④	⑤	⑥
Base	3881.5	3867.6	3896.6	**3249.4**	3556.1	4503.6	3893.5
Base + ③	3249.4	3119.9	3111.7	–	**2386.5**	3161.5	3161.3
Base + ③ + ④	2386.5	**1908.7**	2363.5	–	–	2523.8	2313.8
Base + ① + ③ + ④	1908.7	–	**1768.1**	–	–	2070.0	$\boxed{1796.9}$
Base + ① + ③ + ④ + ⑥	1796.9	–	1858.2	–	–	2256.3	–
Base + all inequalities	1958.8	–	–	–	–	–	–

Table 7.7: Average time (seconds)

Cutting–plane version		①	②	③	④	⑤	⑥
Base	188945.7	198012.9	197918.8	**201891.0**	199566.2	189324.5	194434.5
Base + ③	201891.0	202675.1	201836.4	–	**204795.6**	201349.5	202140.9
Base + ③ + ④	204795.6	**206019.5**	205398.0	–	–	204534.0	205327.6
Base + ① + ③ + ④	206019.5	–	205972.9	–	–	205683.2	$\boxed{\mathbf{206354.2}}$
Base + ① + ③ + ④ + ⑥	**206354.2**	–	206238.3	–	–	205826.9	–
Base + all inequalities	205734.9	–	–	–	–	–	–

Table 7.8: Average lower bound

column of the tables describes the initial configuration used for each batch of runs. The second column reports the results obtained with this inital configuration, while the following ones show the results after adding each one of the remaining separation procedures. As can be seen in these tables, the best overall configuration is the one including all the separation algorithms except for the parity inequalities for sets of one single vertex and the q-connectivity inequalities. Regarding the single vertex parity inequalities, the results are not surprising, since these inequalities are already used in the base version as lazy constraints and, moreover, they can also be found by the more general parity separation algorithm. As for the disappointing performance of the (non-aggregate) q-connectivity inequalities, this can be due to the large number of inequalities of this type found and added by the separation algorithm, which can increase the size of the LP relaxations dramatically. This makes the LP problems harder and prevent the overall algorithm from exploring more nodes of the branch–and–cut tree.

For the final experiments in the next section, we use the configuration providing the best results, that is, the one containing connectivity, general parity, aggregate parity, and aggregate q–connectivity inequalities.

7.6.2 Performance of the branch–and–cut algorithm

Now we present the results obtained with the branch–and–cut algorithm on all the instances described in Section 7.5 with a time limit of 2 hours. These results are reported in Tables 7.9 and 7.10. The first one shows the results grouped by the number of required vertices of the instances, while in the second one the instances are grouped

$\lvert V_R \rvert$	♯ Opt	Gap0 (%)	Gap (%)	Nodes	Time
2	14/14	1.11	0.00	64.9	14.6
3	14/14	1.41	0.00	156.8	52.2
4	13/14	1.98	0.04	1234.7	738.7
5	10/14	5.64	1.51	3467.1	3113.9
6	9/14	5.86	1.27	5009.7	3595.0
7	6/14	9.68	5.40	5488.8	4442.3
	66/84		**1.28**		

Table 7.9: Computational results (instances grouped by number of required vertices)

Grid size	♯ Opt	Gap0 (%)	Gap (%)	Nodes	Time
4×4	12/12	4.17	0.00	317.1	8.2
5×5	12/12	3.82	0.00	2094.8	206.0
6×6	11/12	4.31	0.51	3069.7	895.1
7×7	10/12	3.86	1.02	3737.2	1756.3
8×8	8/12	5.03	2.51	4566.5	3519.3
9×9	8/12	3.05	1.36	2042.1	3161.8
10×10	5/12	4.94	3.56	2125.0	4332.3
	66/84		**1.28**		

Table 7.10: Computational results (instances grouped by size)

according to the size of the grid, since both parameters affect the number of variables in the model directly. In both tables, column '♯ Opt' gives the number of instances solved to optimality within the time limit. Column 'Gap0 (%)' shows the average percentage gap between the final upper bound and the lower bound at the end of the root node, while 'Gap (%)' presents the average percentage gap between the final upper and lower bounds. The last two columns, 'Nodes' and 'Time', present the average number of nodes explored in the branch–and–cut tree and the average computing time in seconds, respectively. The last row of the tables gives the total number of instances solved and the overall average gap.

Overall, it can be concluded that the performance of the algorithm is outstanding. It is able to solve 66 out of 84 instances, with an average gap of 1.28%. If we look at table 7.9, all of the instances with 2 and 3 required vertices are solved, as well as all but one of the instances with 4 required vertices. Instances seem to become a bit harder with 5 or more required vertices, but still 25 out 42 of these instances have been solved optimally. A similar behavior can be observed with respect to the size of the grids. 45 out of 48 instances with sizes from 4×4 to 7×7 have been solved, compared to 31 out of 48 for the larger ones.

Although both the number of required vertices and the size of the grid seem to have a similar impact on the overall difficulty of the instance, both in terms of number of instances solved and average final gap, it is interesting to note the difference with respect to the gap at the end of the root node, ‘Gap0’. While this gap clearly increases as the number of required vertices grows, it seems that it remains more or less constant with the grid size. This seems to indicate that the quality of the lower bound obtained at the root node does not depend on the size of the graph as much as on the number of paths.

Conclusions

Throughout this thesis, we have studied three mathematical optimization problems that arise as an extension of arc routing problems in which drones are used to perform a given service. Arc routing problems with drones differ from classical ARPs in that drones can fly directly between any two points of the network without following the edges of the graph. This enables Drone ARPs to have lower cost solutions than Postman ARPs, but also makes them more difficult to solve since they are continuous optimization problems with an uncountable number of feasible solutions.

The first two chapters of this thesis have introduced the reader to some basic concepts of graph theory, polyhedral combinatorics, and linear and integer programming, as well as offered an overview of some important routing problems that have been studied in depth in the literature.

In Chapter 3, we have introduced and studied the length constrained K–drones rural postman problem (LC K–DRPP), an extension of the Drone RPP introduced in Campbell et al. (2018), in which the limited capacity of the drones makes it impossible to service all the lines requiring service with a single drone and, therefore, we have to find a set of drone routes, each of limited length. We have presented a mathematical formulation for its discrete approximation, the length constrained K–vehicles rural postman problem (LC K–RPP), and two solution methods, a branch–and–cut algorithm for the LC K–RPP and a matheuristic for the original continuous problem. The B&C algorithm is based on the proposed formulation and on the strengthening of its linear relaxation through the separation of several families of valid inequalities that are exponential in number. The matheuristic consists of several features, including edge splitting to increase flexibility in the service, local search to improve solutions, and an optimization procedure that exactly solves RPP instances associated with each single drone route. Both algorithms have been tested on a large set of instances from the literature and on a new set of larger instances with up to 137 lines and with 2 to 6 drones.

In Chapter 4, we have presented a new formulation for the LC K–RPP with two binary variables for each edge and each drone representing the first and second traversals of the edge, respectively. We have studied the polyhedron associated with the set of solutions of a relaxed formulation and proved that several families of inequalities induce facets of this polyhedron. Based on this new formulation and the families of valid inequalities presented, we have designed and implemented in Chapter 5 a branch–and–cut algorithm for the LC K–RPP that incorporates the separation of these inequalities. This branch and cut is the main routine of an iterative algorithm that, by solving an LC K–RPP instance at each step, finds good solutions for the original LC K–DRPP.

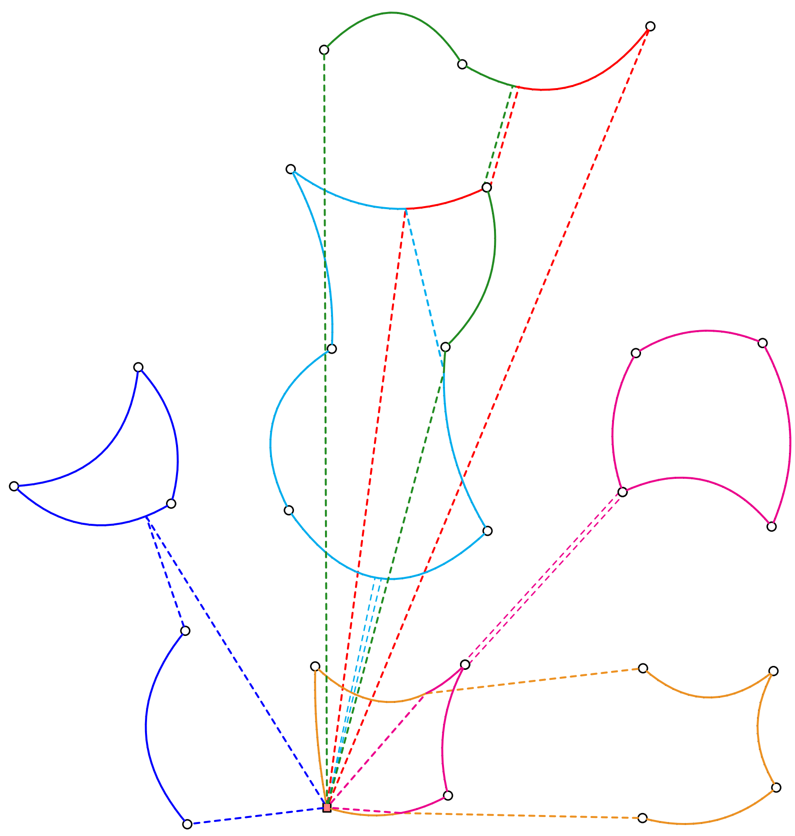


Figure 7.10: Solution of an even DroneRPP66 instance with $L = 1300$ (6 drones)

The computational results obtained show the efficiency of the proposed method.

Figure 7.10 shows an LC K–DRPP solution provided by the matheuristic for the even instance DroneRPP66 with six drones, in which solid lines represent the traversal and service of the required lines and dashed lines are associated with deadheading traversals. This solution has a deadheading cost of 3411, while 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 Figure 7.10 show some of the complexity of these problems and solutions, as well as the benefits the drones get from being able to fly directly between two points. Observe that the green route in Figure 7.10 includes parts of two required lines and all of two other required lines, which are in two separate components of the network. These solutions also reflect the strategy of splitting the arcs in equidistant segments, while a different set of intermediate points may lead to (likely small) improvements. Future research in this line could include 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.

In chapter 6, we have addressed the multi–purpose K–drones general routing problem (MP K–DGRP), where a fleet of multi–purpose drones are used to jointly provide

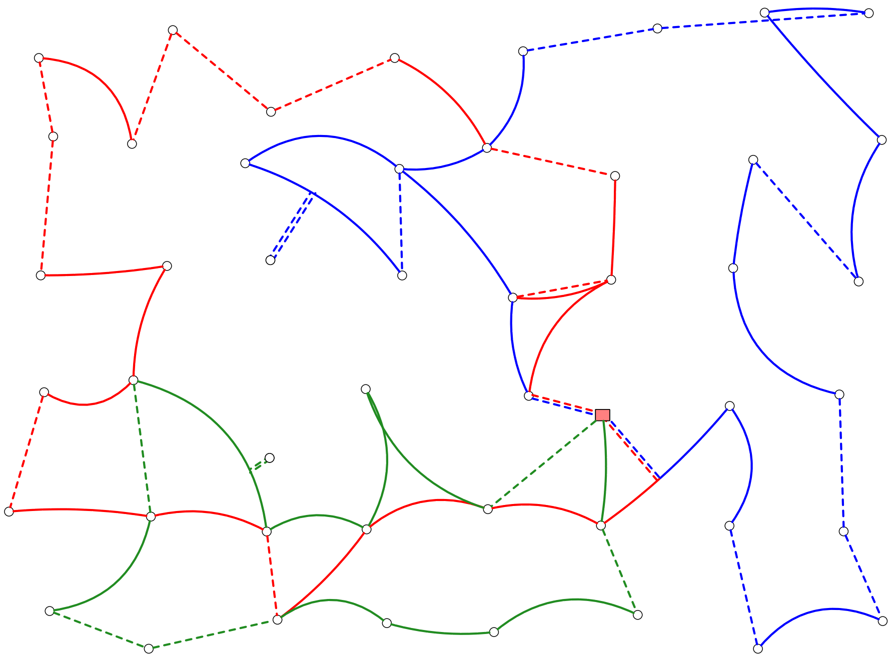


Figure 7.11: Solution of an instance with 8 required vertices and 6 connected components induced by 37 required edges corresponding to a real network.

sensing over a region along with deliveries to discrete nodes. The continuous areas for sensing coverage can be modeled as a set of parallel lines so that each area is completely serviced if all its lines are traversed. For this problem, we have presented a matheuristic algorithm, and we also have proposed a formulation for its discretized version, the K–vehicles GRP, and presented a branch–and–cut procedure for its solution. The results obtained with the branch–and–cut show that the formulation and the valid inequalities are useful for optimally solving small and medium–size K–GRP instances, and provide good lower bounds for larger instances that allow us to measure the quality of the feasible solutions provided by the matheuristic. The matheuristic algorithm is capable of finding very good MP K–DGRP solutions in short computing times.

The solution methods described in chapter 6 can be adapted to deal with general networks associated with required vertices and connected components induced by the required edges (rather than sets of parallel required lines). An example of such a network and a solution to it with three multi–purpose drones can be seen in Figure 7.11. This includes a connected component with 20 required edges serviced by the three drones. We can extend the work by developing versions of both methods that exploit the particularities of these general networks.

Future research in this line could consider a variety of practical extensions for multi–purpose drones. One area that applies for some delivery settings is to prioritize either deliveries or sensing. Thus, one could enforce a requirement to complete deliveries first (before sensing activities) or vice versa. One might also enforce a time constraint on the last delivery, as can be important for medical supplies or perishable products. Another extension could be to allow drone recharging, so the drone does not need to return to the base, but could be recharged (or reloaded for more deliveries) in the field, or to consider several depots from which to launch the drones to joinly perform the service. Research could also explore the optimal orientation of lines to cover a region, or explore other drone flight paths for covering a region. From an algorithmic perspective, future research can address other ways of identifying intermediate points where drones can enter and exit lines, and other heuristic solution approaches for very large problems.

In chapter 7, we have introduced a new combinatorial optimization problem, the load–dependent drone general routing problem (LDdGRP). This problem is a variant of the classical GRP, in which a drone has to traverse some required edges of a graph and also visit a set of nodes that require a delivery. In our problem, the time of traversing each edge of the network is given by the product of the distance travelled and the total weight carried by the drone. For this difficult problem, we have proposed a mathematical formulation and studied its associated polyhedron, and several families of valid inequalities have also been presented. We have designed a branch and cut to solve the LDdGRP that incorporates separation procedures for all these inequalities. The computational experiments carried out reflect the efficiency of our exact approach on a set of instances with up to 7 deliveries and up to 96 required edges.

Regarding this problem, we have approached it from a theoretical point of view, developing an exact procedure for its solution. Given the great difficulty of the problem, we could design and implement heuristic algorithms to solve large LDdGRP instances in reasonable computational times. We could also extend our research to consider a fleet of multi–purpose drones with limited capacity and autonomy to jointly perform all the service. The issue of considering intermediate points on the lines to improve solutions, as done for the LC K–DRPP and the MP K–DGRP, remains open to be studied.

Appendix A

Resumen extendido

La tecnología emergente de vehículos aéreos no tripulados, comúnmente conocidos como *drones*, ha brindado nuevas oportunidades para los profesionales de la logística urbana en la última década. El transporte ha jugado siempre un papel crucial en la sociedad y en la economía, y un motor fundamental del desarrollo económico en los últimos tiempos ha sido la inversión en sistemas de transporte cada vez más eficientes. Los drones presentan ventajas atractivas en comparación con los vehículos terrestres estándar, como evitar la congestión en las redes viales, eliminar el riesgo del personal en operaciones de difícil acceso u obtener una mayor precisión de medición en la inspección de infraestructuras. Muchas empresas comerciales han mostrado recientemente interés en utilizar drones para realizar entregas de última milla más rentables y rápidas. Amazon anunció a finales de 2013 que entregaría paquetes directamente en cada puerta a través de Prime Air usando pequeños drones 30 minutos después de que los clientes presionaran el botón "comprar". Unos años más tarde, lanzaría una versión de su dron de entrega Prime Air que era una aeronave híbrida robusta capaz de despegar y aterrizar verticalmente que podía volar hasta 15 millas y entregar paquetes de menos de cinco libras a los clientes en menos de 30 minutos. Junto con Amazon, otros servicios de entrega como UPS o Google han estado probando el uso potencial de drones para la entrega de paquetes. Dado que los drones aéreos no están restringidos por la infraestructura local, también se pueden utilizar de manera rentable en la distribución rural, la vigilancia y la intralogística, así como en el mapeo geológico y ambiental en 3D para la recopilación de datos. El uso de drones dentro de todos estos escenarios enfrenta múltiples problemas (y desafíos) que pueden ser abordados mediante problemas de rutas, cuyos modelos de solución apuntan a encontrar la ruta (o rutas) más eficiente relacionada con un recurso explícito como la distancia, el tiempo o la energía.

Los problemas de rutas de vehículos se encuentran entre los problemas más estudiados dentro del área de la optimización combinatoria. Dado un vehículo (o una flota de vehículos) y un conjunto de demandas de transporte, el objetivo de estos problemas es determinar una ruta (o conjunto de rutas) de coste mínimo que satisfaga todas las solicitudes o demandas requeridas y cumpla ciertas condiciones adicionales. Muchas situaciones del mundo real se pueden modelar como problemas de rutas. Por ejemplo, la distribución de productos o bienes a clientes, la recolección de basura en las calles, o la inspección y el mantenimiento de carreteras o infraestructuras eléctricas. Todos estos problemas de rutas se modelan matemáticamente representando la red de transporte en un grafo, donde cada uno de sus arcos o aristas representa una conexión (calle, camino, frontera, línea de costa) de la red real y tiene un peso asociado que representa la distancia o el coste de recorrerlo.

Los problemas de rutas clásicos se pueden clasificar de forma natural en dos grupos, problemas de rutas por nodos y problemas de rutas por arcos, según en qué parte del grafo se encuentre el servicio a realizar. En los problemas de rutas por nodos, el servicio se ubica en los vértices del grafo. Estos problemas representarían, por ejemplo, una situación real en la que un viajero debe visitar una serie de ciudades de tal manera que se minimice la distancia recorrida. Entre los problemas de rutas por nodos más conocidos se encuentran el problema del viajante de comercio (*traveling salesman problem*, TSP), cuyo objetivo es encontrar una ruta para un único vehículo que visite todos los nodos de un grafo completo dado, y el problema de rutas de vehículos (*vehicle routing problem*, VRP), donde se dispone de flota de vehículos situados en un depósito y se pretende encontrar una ruta para cada vehículo que satisfaga ciertas limitaciones y de manera que conjuntamente se sirva a un conjunto de clientes dado. Si consideramos, en cambio, la situación en la que un cartero sale de su oficina, entrega el correo por las calles, y luego regresa a su oficina, el diseño de una ruta de longitud mínima para dicho cartero es equivalente en este caso a encontrar un recorrido en un grafo donde algunos arcos (carreteras) tienen una cierta demanda que debe ser satisfecha. A estos problemas de rutas en los que la solución debe atravesar un conjunto de aristas y/o arcos del grafo los llamamos problemas de rutas por arcos (ARPs). En esta tesis, vamos a enfocar nuestro estudio en esta segunda familia de problemas.

El origen de los problemas de rutas por arcos se atribuye al conocido problema de los puentes de Königsberg. En el siglo XVIII, el río Pregel dividía la ciudad prusiana de Königsberg en cuatro partes que estaban conectadas por siete puentes, y popularmente se preguntaba si era posible encontrar una ruta que cruzara cada puente exactamente una vez y luego regresara al punto de partida. Euler argumentó en 1736 que no era posible encontrar dicho recorrido, y el estudio que realizó para abordar este problema sentaría las bases de la teoría de grafos moderna. No fue hasta 1960 que apareció la primera publicación relacionada con un problema de rutas por arcos, el conocido problema del cartero chino. Guan planteó en 1962 el problema de diseñar la ruta de menor distancia para un cartero que tenía que caminar por todas las calles de un barrio determinado para entregar el correo, y propuso un algoritmo (no polinómico) para resolverlo. Unos años más tarde, se demostró que este problema se podía resolver en tiempo polinómico en algunos tipos de grafo. A principios de la década de 1980, aparece una primera descripción general de los problemas de rutas por arcos y se ofrece una clasificación más detallada de estos problemas. Desde entonces, el estudio de problemas de rutas por arcos ha evolucionado mucho dentro del área de la optimización combinatoria, y se han desarrollado una gran variedad de algoritmos exactos y heurísticos que alcanzan un alto nivel de sofisticación, permitiendo la solución de instancias cada vez más grandes en tiempos computacionales muy razonables. El creciente interés por el estudio de este tipo de problemas ha venido motivado, además de por su gran atractivo teórico, por la gran cantidad de situaciones de la vida real que modelan, como la lectura de contadores, el reparto de periódicos, la limpieza de nieve y esparcimiento de sal para el mantenimiento de las carreteras en invierno, la gestión y recogida de residuos en las calles, la inspección de infraestructuras, etc.

Entre los problemas de rutas por arcos más conocidos se encuentran el problema del cartero chino (*Chinese postman problem*, CPP) y el problema del cartero rural (*rural postman problem*, RPP), donde un único vehículo tiene que atravesar todos o algunos de los enlaces del grafo, respectivamente, y el problema de rutas por arcos con capacidades (*capacitated arc routing problem*, CARP), donde se dispone de una flota de vehículos con capacidad limitada para dar servicio conjunto a todos los enlaces requeridos. Todos estos problemas se pueden formular como problemas de optimización combinatoria, donde el objetivo es encontrar, entre un conjunto contable (pero enorme) de soluciones factibles, una que minimice (o maximice) una función de coste, llamada función objetivo. Desafortunadamente, a excepción de algunos de los problemas más simples, no es razonable esperar encontrar algoritmos capaces de resolver cualquier instancia de un problema en un número de operaciones que crece de forma polinómica con el tamaño de la instancia.

El uso de drones para realizar el servicio en los problemas de rutas por arcos implica cambios significativos en la forma tradicional de modelar y resolver estos problemas. En los problemas de rutas por arcos clásicos con vehículos terrestres, las calles que requieren recolección de residuos, los caminos en los que se debe quitar la nieve o las tuberías que se deben inspeccionar, por ejemplo, están representados por aristas o arcos de una red que ignoran la forma geométrica de la infraestructura modelada (aunque no su coste o distancia), ya que estos vehículos tienen que recorrer un arco o arista (un camino) de un extremo al otro. Además, los vehículos en problemas de rutas por arcos tradicionales no pueden viajar fuera de la red. En cambio, los drones aéreos tienen la capacidad de viajar directamente entre dos puntos cualquiera de la red, no necesariamente entre los vértices del grafo. Ellos pueden iniciar el servicio de una arista en el punto más adecuado de la misma según la trayectoria del dron y la forma de cada línea a dar servicio. Esta consideración hace que los problemas de rutas por de arcos con drones sean problemas de optimización continua con un número infinito e incontable de soluciones factibles. Una manera de enfocar estos problemas para poder utilizar la metodología de los problemas de optimización combinatoria se basa en aproximar cada línea o curva en el plano por una cadena poligonal con un número finito de segmentos, y resolver el problema como un problema de optimización discreta, donde se permite a los vehículos entrar y salir de cada línea curva sólo en los puntos de la cadena poligonal. Una vez discretizado, el conjunto de aristas no requeridas de la instancia resultante forma un grafo completo, y el coste de viajar entre cualquier par de puntos de esta red viene dado por la distancia euclídea.

Entre las aplicaciones de problemas de rutas por arcos con drones se incluyen la inspección y el monitoreo de infraestructuras e instalaciones que pueden modelarse como redes o conjuntos de líneas. Dos áreas de aplicación relevantes son los sistemas de transmisión de energía y el transporte. Podemos encontrar ejemplos de aplicaciones de inspección con drones en una amplia variedad de áreas, incluidas líneas eléctricas, vías férreas, alcantarillas, características geográficas, edificios, puentes y turbinas eólicas. En algunas áreas, como las líneas eléctricas, el uso de drones proporciona un coste efectivo, inspecciones más rápidas y seguras. La investigación académica sobre la inspección de líneas eléctricas con drones aún es limitada. Algunos autores modelan la inspección de líneas eléctricas, representadas como segmentos de línea recta, por varios drones que se lanzan desde vehículos terrestres en un conjunto de nodos en la red vial, proponiendo heurísticas constructivas y de mejora para diseñar la planificación de rutas de dicho sistema cooperativo. Otros temas de inspección de drones relacionados con la energía que reciben atención son la supervisión de turbinas eólicas, especialmente para parques eólicos marinos costosos y de difícil acceso, y el monitoreo de instalaciones de petróleo y gas en alta mar y tuberías. En los sistemas de transporte, los ejemplos de problemas de rutas por arcos drones surgen en áreas como el monitoreo del tráfico vial, la inspección de vías férreas y de tránsito, la inspección de puentes o carreteras y la gestión de la vegetación. El monitoreo del tráfico urbano y los sistemas viales proporcionan aplicaciones importantes para las rutas por arco de drones a lo largo de infraestructuras que, naturalmente, se pueden modelar con líneas curvas. Los puentes y otros edificios asociados con los sistemas de transporte también brindan oportunidades para el enrutamiento por arco de drones para garantizar inspecciones eficientes. Otras aplicaciones con vuelos de drones que cubren características lineales incluyen la vigilancia a lo largo de las fronteras y la vigilancia para evitar el ingreso marino cerca de las plantas de energía nuclear, en cuyo caso el área a inspeccionar por los drones se puede representar como una red formada por un conjunto de líneas (que proporcionan cobertura de sensor de las áreas deseadas).

A lo largo de esta tesis, vamos a estudiar tres variantes de problemas de rutas por arcos con drones, que serán modelados como problemas de optimización combinatoria y abordados con procedimientos exactos y heurísticos. La tesis está estructurada en siete capítulos. Los dos primeros son capítulos introductorios destinados a acercar al lector a conceptos y resultados esenciales relacionados con el contenido de la tesis. En el Capítulo 1, se presentan varias definiciones básicas y resultados de teoría de grafos, programación lineal y entera, combinatoria poliédrica y optimización combinatoria, mientras que el Capítulo 2 ofrece una descripción general de los principales problemas de rutas que se han estudiado en la literatura académica.

En los tres capítulos siguientes se aborda el *length constrained K–drones rural postman problem*, o problema del cartero rural con K drones de autonomía restringida (LC K–DRPP). Este problema de optimización continua es una extensión del problema del cartero rural con un dron (o Drone RPP) ya estudiado en la literatura, en el que la autonomía limitada de los drones imposibilita atravesar todas las líneas que requieren servicio con un solo dron y, por lo tanto, tenemos que encontrar un conjunto de rutas para una flota de K drones, cada una de longitud limitada, que conjuntamente sirvan (atraviesen) el conjunto de líneas (curvas o rectas) requeridas de una red.

En el capítulo 3, se definen tanto el LC K–DRPP como su aproximación discreta, el *length constrained K–vehicles rural postman problem* (LC K–RPP). Para este problema de optimización combinatoria presentamos una formulación y algunas desigualdades válidas, así como un algoritmo de ramificación y corte (*branch and cut*) para resolver el problema. También describimos un algoritmo matheurístico para resolver el problema continuo original. El algoritmo de ramificación y corte se basa en la formulación propuesta y en el fortalecimiento de su relajación lineal mediante la separación de varias familias de desigualdades. El algoritmo matheurístico consta de varios procedimientos, que incluyen la división de las líneas de la red en pequeños segmentos para ofrecer más flexibilidad en el servicio de cada línea, varios métodos de búsqueda local para mejorar las soluciones y un procedimiento de optimización que resuelve exactamente las instancias de un problema del cartero rural asociadas con cada ruta de drones. Ambos algoritmos se prueban en un gran conjunto de instancias de la literatura y en un nuevo conjunto de instancias más grandes generado para este trabajo con hasta 137 líneas y con 2 a 6 drones.

El Capítulo 4 profundiza en el estudio del LC K–RPP. En este capítulo proponemos una nueva formulación para este problema de rutas por arcos discreto que considera dos variables binarias para cada arista representando la primera y segunda vez, respectivamente, que ésta es atravesada por el dron en la solución. A lo largo de este capítulo se desarrolla un estudio poliédrico del conjunto de soluciones de una formulación relajada, demostrando que algunas desigualdades de la formulación definen facetas de su poliedro asociado. Se presentan además varias familias de desigualdades válidas y se estudian algunas condiciones bajo las cuales tales desigualdades inducen facetas del poliedro. En el Capítulo 5, se propone un nuevo algoritmo para resolver el LC K–DRPP. Por un lado, presentamos un nuevo algoritmo de ramificación y corte para el LC K–RPP basado en la formulación introducida en el capítulo anterior, que incorpora la separación de las desigualdades válidas propuestas. Por otro lado, este algoritmo de ramificación y corte es la rutina principal de un algoritmo iterativo que resuelve en cada paso una instancia del LC K–RPP para encontrar buenas soluciones para la instancia del problema continuo original. Los resultados computacionales obtenidos muestran la efectividad de este nuevo algoritmo.

Las soluciones obtenidas para el LC K–DRPP con los procedimientos detallados en los capítulos anteriores reflejan que la estrategia de dividir las líneas originales de las instancias en segmentos equidistantes a ser servidos por los drones claramente proporciona costes más reducidos. Un conjunto diferente de puntos intermedios puede conducir a mejoras (probablemente pequeñas). La investigación futura en esta línea podría incluir la exploración de algunas mejoras locales de post–procesamiento para agregar selectivamente más nodos intermedios a ciertas líneas. Algunos nodos interesantes a considerar podrían ser los puntos interiores en un borde requerido que están más cerca de los puntos finales de otros bordes requeridos o/y los puntos en un borde requerido que están más cerca de cualquier punto en otros bordes requeridos.

Los problemas de rutas por arcos y los problemas de rutas por nodos se pueden unificar en los llamados problemas generales de rutas, que son aquellos en los que la demanda de servicio se encuentra tanto en los enlaces como en los nodos del grafo. En los capítulos 6 y 7 de la tesis abordamos dos extensiones del problema general de rutas introducido por Orloff en 1974, donde los vehículos que realicen el servicio van a ser drones multipropósito. Hoy en día, los drones comerciales se implementan para una amplia variedad de misiones, siendo dos grandes e importantes categorías la entrega de bienes a determinados clientes y la detección e inspección. La obtención de rutas óptimas para drones cuyos vuelos combinen entrega de paquetes y detección (mapeo) en un solo viaje es el tema clave que abordamos aquí.

En el Capítulo 6, presentamos y estudiamos el problema general de rutas para K drones multipropósito (MP K–DGRP). En este problema de optimización continua, una flota de drones multipropósito, vehículos aéreos que pueden realizar entregas y realizar actividades de inspección, deben visitar conjuntamente un conjunto de nodos para realizar entregas y también mapear una o varias áreas más continuas. Una manera de cubrir superficies bidimensionales es reemplazar cada región de interés por un conjunto de líneas paralelas que seguirán los drones para proporcionar una cobertura completa del área. Por lo general, las líneas paralelas están orientadas para minimizar la cantidad de giros del dron, ya que los giros interrumpen la recopilación de datos y aumentan el tiempo de vuelo del dron y el uso de la batería. Dado que permitimos que se cubra una región en varios viajes diferentes de drones, la orientación óptima de los vuelos puede ser diferente para diferentes partes de la región. Adoptamos aquí la estrategia razonable para cubrir una región en la que los drones vuelan siguiendo líneas paralelas orientadas con el eje más largo de la región (minimizando así los giros que necesitan los drones), donde el espaciado de las líneas refleja las características del sensor que están utilizando los drones. Aunque encontrar la forma óptima de diseñar estas líneas paralelas es un problema interesante y difícil en sí mismo, en este trabajo suponemos que las líneas que deben seguir los drones para proporcionar detección sobre las áreas son paralelas, rectas y ya dadas.

Dado así un conjunto de líneas que cubren las áreas a mapear y un conjunto de puntos con una determinada demanda, el problema que estudiamos consiste en diseñar rutas de drones de duración total mínima, recorriendo conjuntamente todas las líneas y visitando todos los puntos (para hacer entregas). La duración de cada ruta no puede exceder un límite de tiempo y la demanda entregada por cada ruta no puede exceder la capacidad del dron. Como en el LC K–DRPP estudiado en capítulos previos, los drones pueden viajar fuera de la red, entrando y saliendo de cada línea dada en cualquier punto de la misma, por lo que el MP K–DGRP es un problema de optimización continua. En el capítulo 6 proponemos un algoritmo matemático para resolver dicho problema. Además, definimos el problema general de rutas para K vehículos multipropósito (MP K–GRP), un problema de optimización discreto cuya solución proporciona un límite superior de la solución óptima del problema original. Para este problema de optimización combinatoria, presentamos una formulación matemática con variables binarias y varias familias de desigualdades válidas, y proponemos un algoritmo de ramificación y corte para resolverlo. Los extensos experimentos computacionales realizados en dos conjuntos de instancias generadas de manera aleatoria para probar el rendimiento de los algoritmos propuestos se muestran al final del capítulo.

En el problema general de rutas se asume que el coste de viajar desde un nodo i a un nodo j en el grafo es una constante c_{ij}. No obstante, el coste real de un vehículo que se desplaza entre cualquier par de nodos puede depender de muchos otros aspectos, como la carga que lleva el vehículo y el consumo de combustible o energía. En los sistemas de transporte con drones, el peso de la carga transportada representa una parte significativa del peso bruto del vehículo y puede influir decisivamente en el consumo de batería y la autonomía de vuelo del dron, así como en los tiempos de recorrido de las aristas de la red y los tiempos de despegue y aterrizaje del dron. Por ello es importante tener en cuenta el peso que lleva el dron en cada momento del recorrido. Tal variación del peso de la carga a lo largo del viaje se tiene en cuenta en el problema que estudiamos en el capítulo 7 para determinar la duración mínima de la ruta del dron. Considerar la dependencia de la carga agrega una dificultad considerable para modelar y resolver problemas de rutas, ya que los costos y los tiempos de viaje en la red ya no son constantes.

En el capítulo 7, introducimos y estudiamos el problema general de rutas con un dron con costes dependientes de la carga (LDdGRP). El objetivo de este problema es encontrar un recorrido de duración mínima para un dron con capacidad de detección y entrega que, comenzando y terminando en un depósito determinado, atraviesa un conjunto de aristas requeridas de una red y también entrega mercancías en un conjunto de nodos requeridos. En el LDdGRP se asume que el tiempo de recorrido de cada arista, así como los tiempos de despegue y aterrizaje, son proporcionales al producto de la distancia recorrida y el peso total (incluida la carga) del dron. Como en otros problemas de rutas por arcos estudiados en capítulos anteriores, los drones pueden viajar en línea recta entre dos puntos cualesquiera de la red, y el problema de optimización es continuo y muy difícil de resolver. Estos problemas se pueden discretizar aproximando cada línea original de la red por uno o varios segmentos (aristas) requeridos y permitiendo que los drones entren y salgan de estos bordes solo en sus puntos finales. Las características originales de los problemas de enrutamiento de arco de drones que permanecen en el problema discreto son que el conjunto de aristas no requeridas induce un grafo completo (por lo tanto, cada arista requerida tiene una paralela no requerida) y que los tiempos de atravesar cada arista no requerida de la red satisfacen la desigualdad triangular. A diferencia de otros capítulos, aquí nos centramos en formular el problema discreto y proponemos un algoritmo exacto de ramificación y corte para su solución, sin abordar el problema de incluir y eliminar puntos intermedios sobre las rectas originales para mejorar la solución del problema continuo.

Finalmente, la tesis incluye una sección de conclusiones que sirve de recapitulación de todo el trabajo desarrollado, donde se marcan además algunas líneas de trabajo futuro. Parte del contenido de esta tesis ha sido publicado o enviado para su publicación a revistas internacionales bien posicionadas del área. El contenido del capítulo 3 se basa en el siguiente artículo publicado:

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

Parte del estudio poliédrico desarrollado en el capítulo 4 para el problema del cartero rural con K drones con autonomía limitada (el caso especial $K = 1$) se puede encontrar en

- ▷ Á. Corberán, I. Plana, J. M. Sanchis, and P. Segura (2021). Polyhedral study of a new formulation for the Rural Postman Problem. *Technical report*, http://www.uv.es/plani/reports.html

y una versión más compacta del contenido de los capítulos 4 y 5 está publicado en el siguiente artículo:

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

El capítulo 6 está basado en el siguiente artículo que ya ha sido enviado a una revista internacional para su publicación

- ▷ J. F. Campbell, Á. Corberán, I. Plana, J. M. Sanchis, and P. Segura (2023). The multi–purpose K drones general routing problem. *Under revision.*

El contenido del capítulo 7 será enviado pronto a una revista reconocida también en el área.

Bibliography

[1] Aerolab (2022). XAG P40 Multipurpose Spray Drone (20L), https://www.aerolab.co.nz/collections/agriculturaldrones/products/xagp40.

[2] Agatz, N., Bouman, P., and Schmidt, M. (2018). Optimization approaches for the traveling salesman problem with drone. *Transport. Sci.* 52 (4), 739–1034.

[3] Aminu U. F. and Eglese R. W. (2006). A constraint programming approach to the Chinese postman problem with time windows. *Computers & Operations Research*, 33, 3423–3431.

[4] Amorosi, L., Puerto, J., and Valverde, C. (2021). Coordinating drones with mothership vehicles: the mothership and drone routing problem with graphs. *Comput. Operations Res.* 136. https://doi.org/10.1016/j.cor.2021.105445

[5] Applegate, D. L., Bixby, R. E., Chvatál, V., and Cook, W. J. (2006). The Traveling Salesman Problem: A Computational Study. Princeton University Press.

[6] Archetti, C., Bertazzi, L., Laganà, D., and Vocaturo, F. (2017). The Undirected Capacitated General Routing Problem with Profits. *European Journal of Operational Research* 257, 822–833.

[7] Assad, A., Ball, M., Bodin, L., and Golden, B. (1983). Routing and scheduling of vehicles and crews. *Computers & Operations Research* 10, 63–211.

[8] Ávila, T., Corberán, Á., Plana, I., Sanchis, J. M. (2015). A new branch–and–cut algorithm for the generalized directed rural postman problem. *Transportation Science* 50 (2), 750–761.

[9] Bach, L., Hasle, G., and Wøhlk, S. (2013). A lower bound for the Node, Edge, and Arc Routing Problem. *Computers & Operations Research* 40, 943–952.

[10] Baldacci, R. and Maniezzo, M. (2006). Exact methods based on node routing formulations for undirected arc routing problems. *Networks* 47, 52–60.

[11] Baldacci, R., Mingozzi, A., and Roberti, R. (2012). Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research* 218, 1–6.

[12] Baldacci, R., Toth, P., and Vigo, D. (2010). Exact algorithms for routing problems under vehicle capacity constraints. *Annals of Operations Research* 175, 213–245.

[13] Baldacci, R., Toth, P., and Vigo, D. (2011). Exact solution of the capacitated vehicle routing problem, in Wiley Encyclopedia in Operations Research and Management Science, vol. 3, Wiley, New York, 2011, 1795–1807.

[14] Balinski, M. L. and Russakoff, A. (1974). On the Assignment Polytope. *SIAM Review* 16 (4), 516–525.

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

[16] Beasley, J. E. (1983). Route First–Cluster Second Methods for Vehicle Routing. *Omega* 11, 403–408.

[17] Bektas T. and Laporte G. (2011). The pollution–routing problem. *Transportation Res. Part B* 45, 1232–1250.

[18] Belenguer, J. M. and Benavent, E. (1994). Branch and cut for the CARP, in Worshop on Algorithmic Approaches to Large and Complex Combinatorial Optimization Problems, Giens, France.

[19] Belenguer, J. M. and Benavent, E. (2003). A cutting plane algorithm for the capacitated arc routing problem. *Computers & Operations Research* 30, 705–728.

[20] Belenguer, J. M., Benavent, E., Lacomme, L., and Prins, C. (2006). Lower and upper bounds for the mixed capacitated arc routing problem. *Computers & Operations Research* 33, 3363–3383.

[21] Benavent, E., Campos, V., Corberán, Á., and Mota, E. (1983). Problemas de rutas por arcos. *Qüestió* 7, 479–490.

[22] Berge, C. (1973). Graphs and Hypergraphs. North Holland, Amsterdam.

[23] Blais, M. and Laporte, G. (2003). Exact solution of the generalized routing problem through graph transformations. *The Journal of the Operational Research Society* 54, 906–910.

[24] Bodin L. and Golden, B. (1981). Classification in vehicle routing and scheduling. *Networks* 11, 97–108.

[25] Bondi, J. A. and Murty, U. S. R. (1976). Graph Theory with Applications. American Elsevier, New York.

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

[27] Braysy, O. and Gendreau, M. (2005). Vehicle routing problem with time windows, part I: Route construction and local search algorithms. *Transportation Science* 39, 104–118.

[28] Campbell, J. F., Corberán, Á., Plana, I., and Sanchis, J. M. (2018). Drone arc routing problems. *Networks* 72, 543–559.

[29] 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/.

[30] CBSNews.com (2013). Amazon unveils futuristic plan: delivery by drone. Available at https://www.cbsnews.com/news/amazon-unveils-futuristic-plan-delivery-by-drone (accessed January 10, 2020).

[31] Christofides, N., Campos, V., Corberán, Á., and Mota, E. (1981). An algorithm for the rural postman problem. Tech. Report IC.O.R.81.5, Imperial College, London.

[32] Christofides, N. (1975). Graph Theory. An Algorithmic Approach. Academic Press, New York.

[33] Clarke, G. and Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research* 12, 568–581.

[34] Coelho, B. N., Coelho, V. N., Coelho, I. M., Ochi, L. S., Haghnazar, K. R., Zuidema, D., Lima, M. S. F., da Costa, A. R. (2017). A multi–objective green UAV routing problem. *Comput. Oper. Res.* 88, 306–315.

[35] Corberán, Á., Eglese, R., Hasle, G., Plana, I., and Sanchis, J. M. (2021). Arc routing problems: a review of the past, present and future. *Networks* 77, 88–115.

[36] Corberán, Á., Erdogan, G., Laporte, G., Plana, I., and Sanchis, J. M. (2018). The Chinese postman problem with load–dependent costs. *Transportation Science* 52 (2), 370–385.

[37] Corberán, Á. and Laporte, G. (editors) (2014). Arc Routing: Problems, Methods, and Applications. *MOS–SIAM Series on Optimization*, Philadelphia.

[38] Corberán, Á., Letchford, A. N., and Sanchis, J. M. (2001). A cutting plane algorithm for the general routing problem. *Mathematical Programming* 90, 291–316.

[39] Corberán, Á., Mejía, G., and Sanchis, J. M. (2005). New results on the mixed general routing problem. *Operations Research* 53 (2), 363–376.

[40] Corberán, Á., Mota, E., and Sanchis, J. M. (2006). A comparison of two different formulations for arc routing problems on mixed graphs. *Computers & Operations Research* 33, 3384–3402.

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

[42] Corberán, Á., Plana, I., and Sanchis, J.M. (2007). A Branch & Cut Algorithm for the Windy General Routing Problem and special cases. *Networks* 49, 245–257.

[43] Corberán, Á., Oswald, M., Plana, I., Reinelt, G., and Sanchis, J. M. (2012). New results on the windy postman problem. *Mathematical Programming* 132, 309–332.

[44] Corberán, Á., Romero, A., and Sanchis, J. M. (2003). The mixed general routing problem polyhedron. *Mathematical Programming* 96 (1), 103–137.

[45] Corberán, Á. and Sanchis, J. M. (1994). A Polyhedral Approach to the Rural Postman Problem. *European Journal of Operational Research* 79, 95–114.

[46] Corberán, Á. and Sanchis, J. M. (1998). The general routing problem polyhedron: Facets from the RPP and GTSP polyhedra. *European Journal of Operational Research* 108, 538–550.

[47] Cordeau, J. F., Laporte, G., Savelsbergh, M. W. P., and Vigo, D. (2007). *Vehicle routing*, in Transportation, Barnhart, C. and Laporte, G., eds., vol. 14 of Handbooks in Operations Research and Management Science, Elsevier, ch. 6, 367–428.

[48] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press and McGraw–Hill.

[49] Cornuejols, G., Fonlupt, J., and Naddef, D. (1985). The traveling salesman problem on a graph and some related inequalities. *Mathematical Programming* 33, 1–27.

[50] Cuong, N. V., Hong, Y. W. P., and Sheu, J. P. (2022). UAV Trajectory Optimization for Joint Relay Communication and Image Surveillance. *IEEE Transactions on Wireless Communications* 21, 10177–10192.

[51] Dabia, S., Demir, E., and Woensel, T.V. (2017). An exact approach for a variant of the pollution–routing problem. *Transportation Science* 51, 607–628.

[52] Dantzig, G. (1963). Linear Programming. Freeman, New York.

[53] Dantzig, G. and Ramser, J. H. (1959). The truck dispatching problem. *Management Science* 6, 80–91.

[54] 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/.

[55] DHL (2014). Unmanned aerial vehicle in logistics, a DHL perspective on implications and use cases for the logistics industry. Available at http://www.dhl.com/ content/dam/downloads/g0/about_us/logistics_insights/DHL_TrendReport_UAV.pdf (accessed January 8, 2020).

[56] Di Puglia Pugliese, L. and Guerriero, F. (2017). Last–mile deliveries by using drones and classical vehicles. In: Sforza, A., Sterle, C. (Eds.), International Conference on Optimization and Decision Science, ODS 2017, Springer Proceedings in Mathematics and Statistics, Springer, New York, 557–565.

[57] DJI Agriculture (2022). What does agricultural drone spraying of 66.7 million hectares mean to the planet?, https://ag.dji.com/newsroom/dji-ag-news-en-greener.

[58] Dorling, K., Heinrichs, J., Messier, G. G., and Magierwski, S. (2017). Vehicle routing problem for drone delivery. *IEEE Trans. Syst., Man, Cybernet.: Syst.* 47 (1), 1–16.

[59] Dror, M. and Haouari, M. (2000). Generalized Steiner problems and other variants. *Journal of Combinatorial Optimization* 4, 415–436.

[60] Dror, M., Stern, H., and Trudeau, P. (1987). Postman tour on a graph with precedence relation on arcs. *Networks* 17, 283–294.

[61] Duarte, A., Mladenovic, N., Sánchez–Oro, J., and Todosijevic, R. (2018). Variable Neighborhood Descent. In: Martí R., Pardalos P., and Resende M. (eds). Handbook of Heuristics. Springer International Publishing AG.

[62] Dukkanci, O., Kara, B. Y., and Bektas, T. (2021). Minimizing energy and cost in range–limited drone deliveries with speed optimization. *Transportation Research Part C* 125, 102985.

[63] Durdevic, P., Ortiz–Arroyo, D., Li, S., and Yang, Z. (2019). Vision aided navigation of a quad–rotor for autonomous wind–farm inspection. *IFAC–PapersOnLine* 52 (8), 61–66.

[64] Edmonds, J. and Johnson, E. (1973). Matching, Euler tours and the Chinese postman. *Mathematical Programming* 5, 88–124.

[65] Eskandaripour, H. and Boldsaikhan, E. (2023). Last–Mile Drone Delivery: Past, Present, and Future. *Drones* 7 (2), 77.

[66] Euler, L. (1936). Solutio problematis ad geometriam situs pertinentis. *Commentarii Academiae Scientiarum Imperialis Petropolitanae* 8, 128–140.

[67] Fleischmann, B. (1985). A cutting plane procedure for the traveling salesman problem on road networks. *European Journal of Operational Research* 21, 147–172.

[68] Fleischmann, B. (1988). A new class of cutting planes for the symmetric traveling salesman problem. *Mathematical Programming* 40, 225–246.

[69] Fontaine, P. (2022). The vehicle routing problem with load–dependent travel times for cargo bicycles. *European Journal of Operational Research* 300, 1005–1016.

[70] Frederickson, G. N., Hecht, M. S., and Kim, C. E. (1978). Approximation algorithms for some routing problems. *SIAM Journal on Computing* 7, 178–193.

[71] Galle, B., Arellano, S., Bobrowski, N., Conde, V., Fischer, T. P., Gerdes, G., Gutmann, A., Hoffmann, T., Itikarai, I., Krejci, T., Liu, E. J., Mulina, K., Nowicki, S., Richardson, T., Rüdiger, J., Wood, K., and Xu, J. (2021). A multi–purpose, multi–rotor drone system for long–range and high–altitude volcanic gas plume measurements. *Atmospheric Measurement Techniques* 14, 4255–4277.

[72] Ghiani, G. and Improta, G. (2000). An algorithm for the hierarchical Chinese postman problem. *Operations Research Letters* 26, 27–32.

[73] Ghiani, G., Laganà, D., and Musmanno, R. (2006). A constructive heuristic for the undirected rural postman problem. *Computers & Operations Research* 33, 3450–3457.

[74] Ghiani, G. and Laporte, G. (2000). A branch–and–cut algorithm for the Undirected Rural Postman Problem. *Mathematical Programming* 87, 467–481.

[75] Golden, B. L., De Armon, J. S., and Baker, E. K. (1983). Computational experiments with algorithms for a class of routing problems. *Computers & Operations Research* 10, 47–59.

[76] Golden, B. L. and Wong, R. T. (1981). Capacitated arc routing problems. *Networks* 11, 305–315.

[77] Gómez–Cabrero, D., Belenguer, J. M., and Benavent, E. (2005). Cutting plane and column generation for the capacitated arc routing problem. Proceedings of the ORP3 2005 (Operational Research Peripathetic Postgraduate Programme), Valencia, Spain, 259–266.

[78] Gouveia, L., Mourao, M. C., and Pinto, S. L. (2010). Lower bounds for the mixed capacitated arc routing problem. *Computers & Operations Research* 37, 692–699.

[79] Groves, G. W. and Van Vuuren, J. H. (2005). Efficient heuristics for the rural postman problem. *ORiON* 21, 33–51.

[80] Guan, M. (1962). Graphic programming using odd and even points. *Chinese Mathematics* 1, 273–277.

[81] Guan, M. (1984). On the windy postman problem. *Discrete Applied Mathematics* 9, 41–46.

[82] Harary, F. (1969). Graph theory. Addison Wesley, Reading, MA.

[83] HD Air Studio (2022). Sonda X8 UAS, Foldable multi–mission octocopter, https://www.unmannedsystemstechnology.com/company/hd-air-studio/sonda-x8-uas/.

[84] Hertz, A., Laporte, G., and Nanchen Hugo, P. (1999). Improvement procedures for the undirected rural postman problem. *INFORMS Journal on Computing* 11, 53–62.

[85] Hierholzer, C. (1873). Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen* 6, 30–32.

[86] Ilavarasi, K. and Joseph, K. S. (2014). Variants of travelling salesman problem: A survey. International Conference on Information Communication and Embedded Systems (ICICES2014), Chennai, India, 1–7.

[87] Jones, D., Gates, A., Huvenne, V., Phillips, A., and Bett, B. (2019). Autonomous marine environmental monitoring: application in decommissioned oil fields. *Sci. Total Environ.* 668, 835–853.

[88] Jordan, S., Moore, J., Hovet, S., Box, J., Perry, J., Kirsche, K., Lewis, D. D., and Tse, Z. (2018). State–of–the–art technologies for UAV inspections. *IET Radar Sonar Navig* 12, 151–164.

[89] Kara, I., Kara, B. Y., and Kadri Yetis, M. (2007). Energy minimizing vehicle routing problem. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 4616, 62–71, LNCS.

[90] Karaduman, M., Çinar, A., and Eren, H. (2019). UAV traffic patrolling via road detection and tracking in anonymous aerial video frames. *J. Intell. Robot Syst.* 95, 675–690.

[91] Karmarkar, N. (1984). A new polynomial–time algorithm for linear programming. *Proceedings of the sixteenth annual ACM symposium on Theory of computing.* ACM.

[92] Karp, R. M. (1972). Reducibility among combinatorial problems. *Springer US*, Boston, MA, 85–103.

[93] Khachian, L. (1979). A polynomial algorithm for linear programming. *Soviet Math. Doklady* 20, 191–194.

[94] Khosravi, M., Enayati, S., Saeedi, H., and Pishro–Nik, H. (2021). Multi–Purpose Drones for Coverage and Transport Applications. *IEEE Transactions on Wireless Communications* 20, 3974–3987.

[95] Klee, V. and Minty, G. (1972). How good is the simplex algorithm. In Shisha, O., editor, *Inequalities III*, 159–175. Academic Press, New York.

[96] Knight, R. (2019). UAV Inspection At The Biggest Oil Rig In The World. December 2, 2019. https://www.microdrones.com/en/content/uav-inspection-at-the-biggest-oil-rig-in-the-world/.

[97] Kumar, N., Ghosh, M., and Singhal, C. (2020). UAV Network for Surveillance of Inaccessible Regions with Zero Blind Spots. IEEE INFOCOM 2020 – IEEE Conference on Computer Communications Workshops, Toronto, Canada, 1213–1218.

[98] Laporte, G. (1992). The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research* 59, 345–358.

[99] Letchford, A. N. (1997). New inequalities for the general routing problem. *European Journal of Operational Research* 96 (2), 317–322.

[100] Letchford, A. N. (1999). The general routing problem: a unifying framework. *European Journal of Operational Research* 112 (1), 122–133.

[101] Lenstra, J. K. and Rinnooy Kan, A. H. (1976). On general routing problems. *Networks* 6, 273–280.

[102] Letchford, A. N., Reinelt, G., and Theis, D. O. (2008). Odd minimum cut–sets and b–matchings revisited. *SIAM J. Discr. Math.* 22, 1480–1487.

[103] Li, M., Zhen, L., Wang, S., Lu, W., and Qu, X. (2018). Unmanned aerial vehicle scheduling problem for traffic monitoring. *Comput. Ind. Eng.* 122, 15–23.

[104] Liu, Y., Shi, J., Liu, Z., Huang, J., and Zhou, T. (2019). Two–layer routing for high–voltage powerline inspection by cooperated ground vehicle and drone. *Energies* 12, 1385.

[105] Luo, H., Zhang, P., Wang, J., Wang, G., and Meng, F. (2019). Traffic patrolling routing problem with drones in an urban road system. *Sensors* 19, 51–64.

[106] Mack, E. (2018). How delivery drones can help save the world. Forbes.com. Available at https://www.forbes.com/sites/ericmack/2018/02/13/delivery-drones-amazon-energy-efficient-reduce-climate-change-pollution/#77130a0a6a87 (accessed January 8, 2020).

[107] McCormack, E. and Stimberis, J. (2010). Small unmanned aircraft evaluated for avalanche control. *Transportation Research Record* 2169, 168–173.

[108] Macrina, G., Di Puglia Pugliese, L., Guerriero, F., and Laporte, G. (2020). Drone–aided routing: A literature review. *Transportation Research Part C: Emerging Technologies* 120, 102762.

[109] Minieka, E. (1979). The chinese postman problem for mixed networks. *Management Science* 25, 643–648.

[110] Mladenovic, M. and Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research* 24, 1097–1100.

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

[112] Murray, C. C. and Chu, A. (2015). The flying sidekick traveling salesman problem: Optimization of drone–assisted parcel delivery. *Transport. Res. Part C: Emerg. Technol.* 54, 86–109.

[113] Nemhauser, G. L. and L.A. Wolsey, L. A. (1988). Integer and Combinatorial Optimization. Wiley–Interscience Series in Discrete Mathematics and Optimization, Wiley, New York.

[114] New, W. K. and Leow, C. Y. (2021). Unmanned Aerial Vehicle (UAV) in Future Communication System. In 26th IEEE Asia-Pacific Conference on Communications (APCC), 217–222.

[115] Orloff, C. S. (1974). A fundamental theorem in vehicle routing. *Networks* 27, 35–64.

[116] Outay, F., Mengash, H. A., and Adnan, M. (2020). Applications of unmanned aerial vehicle (UAV) in road safety, traffic and highway infrastructure management: recent advances and challenges. *Transp. Res. Part A* 141, 116–129.

[117] Padberg, M. W. and Rao, M. R. (1982). Odd minimum cut–sets and b–matchings. *Mathematics for Operations Research* 7, 67–80.

[118] Pandi, R. and Muralidharan, B. (1995). A capacitated general routing problem on mixed networks. *Computers & Operations Research* 22, 465–478.

[119] Papadimitriou, C. (1976). On the complexity of edge traversing. *Journal of the ACM* 23, 544–554.

[120] Papadimitriou, C. (1984). Polytopes and complexity. In Pulleyblank, W., editor, *Progress in Combinatorial Optimization*, 295–305. Academic Press, Toronto.

[121] Parragh, S., Doerner, K., and Hartl, R. (2008a). A survey on pickup and delivery problems part I: Transportation between customers and depot. *Journal für Betriebswirtschaft* 58, 21–51.

[122] Parragh, S., Doerner, K., and Hartl, R. (2008b). A survey on pickup and delivery problems part II: Transportation between pickup and delivery locations. *Journal für Betriebswirtschaft* 58, 81–117

[123] Plotnikov, M., Ni, D., and Price, D. (2019). The Application of Unmanned Aerial Systems In Surface Transportation Volume II–A: development of a Pilot Program to Integrate UAS Technology to Bridge and Rail Inspections. Massachusetts Department of Transportation Report 19–010.

[124] Prins, C., Labadi, N., and Reghioui, M. (2009). Tour splitting algorithms for vehicle routing problems. *International Journal of Production Research* 47 (2), 507–535.

[125] Prins, C., Lacomme, P., and Prodhon, C. (2014). Order–first split–second methods for vehicle routing problems: A review. *Transportation Research Part C: Emerging Technologies* 40, 179–200.

[126] Puerto, J. and Valverde, C. (2022). Routing for unmanned aerial vehicles: Touring dimensional sets. *European Journal of Operational Research* 298, 118–136.

[127] Rauhakallio, P. (2020). The Past, Present, and Future of Powerline Inspection Automation. POWER. https://www.powermag.com/the-past-present-and-future-of-powerline-inspection-automation/.

[128] Rojas Viloria, D., Solano–Charris, E.L., Muñoz–Villamizar, A. and Montoya–Torres, J.R. (2020). Unmanned aerial vehicles/drones in vehicle routing problems: a literature review. Int. Trans. Oper. Res. https://doi.org/10.1111/itor.12783.

[129] Savelsbergh, M. W. P. (1985). Local search in routing problems with time windows. *Ann Oper Res* 4, 285–305.

[130] Shafiee, M., Zhou, Z., Mei, L., Dinmohammadi, F., Karama, J., and Flynn, D. (2021). Unmanned aerial drones for inspection of offshore wind turbines: a mission–critical failure analysis. *Robotics* 10, 26. https://doi.org/10.3390/robotics10010026.

[131] Silvagni, M., Tonoli, A., Zenerino, E., and Chiaberge, M. (2017). Multipurpose UAV for search and rescue operations in mountain avalanche events. *Geomatics, Natural Hazards and Risk* 8, 18–33.

[132] Sujit, P. B., Hudzietz, B. P., and Saripalli, S. (2012). Route planning for angle constrained terrain mapping using an unmanned aerial vehicle. *Journal of Intelligent and Robotic Systems: Theory and Applications* 69, 1–4, 273–283.

[133] Torabbeigi, M., Lim, G. J., and Kim, S. J. (2020). Drone Delivery Scheduling Optimization Considering Payload–induced Battery Consumption Rates. *Journal of Intelligent & Robotic Systems* 97, 471–487.

[134] Toth P. and Vigo D. (2002). Models, relaxations and exact approaches for the capacitated vehicle routing problem. *Discrete Applied Mathematics* 123, 487–512.

[135] Toth P. and Vigo D. (2014). Vehicle routing: problems, methods and applications (Second edition). Society for Industrial and Applied Mathematics: Mathematical Optimization Society, Philadelphia.

[136] Troudi, A., Addouche, S. A., Dellagi, S. and El Mhamedi, A. (2019). Sizing of the drone delivery fleet considering energy autonomy. *Sustainability* 10 (9), 1–17.

[137] Ulmer, M. W. and Thomas, B. W. (2018). Same–day delivery with a heterogeneous fleet of drones and vehicles. *Networks* 72 (4), 475–505.

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

[139] Vasquez–Gomez, J. I., Herrera–Lozada, J. C., and Olguin–Carbajal, M. (2018). Coverage path planning for surveying disjoint areas. In *Proceedings of the International Conference on Unmanned Aircraft Systems (ICUAS).* Dallas, TX, USA, June 2018, 899–904.

[140] Vasquez–Gomez, J. I., Marciano–Melchor, M., Valentin, L., and Herrera–Lozada, J. C. (2020). Coverage Path Planning for 2D Convex Regions. *Journal of Intelligent and Robotic Systems* 97, 81–94.

[141] Vincent, J. and Gartenberg, C. (2019). Here's Amazon's new transforming Prime Air delivery drone. Available at https://www.theverge.com/2019/6/5/18654044/amazon-prime-air-delivery-drone-new-design-safety-transforming-flight-video (accessed January 12, 2020).

[142] Wang, Y., Kirubarajan, T., Tharmarasa, R., Jassemi–Zargani, R., and Kashyap, N. (2018). Multiperiod coverage path planning and scheduling for airborne surveillance. *IEEE Transactions on Aerospace and Electronic Systems* 54, 2257–2273.

[143] Wang, X., Poikonen, S. and Golden, B. L. (2017). The vehicle routing problem with drones: several worst–case results. *Opt. Lett.* 11 (4), 679–697.

[144] Wang, Z. and Sheu, J. B. (2019). Vehicle routing problem with drones. Transport. *Res. Part B: Methodol.* 122, 350–364.

[145] Wishart, J., Lennertz, T., and Hasson, D. (2020). Use Cases for Unmanned Aircraft Systems (UAS) in Public Transportation Systems. Federal Transit Administration FTA Report No. 0176, John A. Volpe National Transportation Systems Center, December 2020.

[146] Win, Z. (1989). On the windy postman problem on Eulerian graphs. *Mathematical Programming* 44, 97–112.

[147] Xie, J., Garcia Carrillo, L. R., and Jin, L. (2020). Path planning for UAV to cover multiple separated convex polygonal regions. *IEEE Access* 8, 51770–51785.

[148] Yang, C. H., Tsai, M. H., Kang, S. C., and Hung, C. Y. (2018). UAV path planning method for digital terrain model reconstruction – a debris fan example. *Automation in Construction* 93, 214–230.

[149] Zachariadis, E. E., Tarantilis, C. D., and Kiranoudis, C. T. (2015). The load–dependent vehicle routing problem and its pick–up and delivery extension. *Transportation Res. Part B* 71, 158–181.

[150] Zhang, J., Campbell, J. F., Sweeney, D. C., and Hupman, A. C. (2021). Energy consumption models for delivery drones: A comparison and assessment. *Transportation Research Part D: Transport and Environment* 90, 102668.

[151] https://medium.com/frontier-technologies-hub/multi-purpose-drone-operation-in-malawi-b4cf671b14d1

[152] Swoop Aero, Strengthening health supply chains in Malawi, https://swoop.aero/solutions/malawi/

[153] https://www.fastcompany.com/90645533/how-drones-are-aiding-in-international-development

[154] https://www.aerosociety.com/news/multirole-lifesavers-a-new-approach-to-humanitarian-drones/