Technical University of Catalonia
Department of Statistics and Operation Research

The Clustered Prize-collecting Arc Routing Problem

By
Carles Franquesa i Niubò

Advisors
Julián Aráoz Durand and Elena Fernández Aréizaga

September 1st, 2008

A la meva germana Marta, in memoriam.

List of errata

- pag. 7 - Terms *size* and *order* are interchanged.
- pag. 102 - Addition of $x()$ to Equation (7.3), leading to

$$x(\delta^+(u)) = x(\delta^-(u)), \qquad u \in V(D) \quad (7.3).$$

- pag. 110 - Mandatory constraint (7.9) has been transformed to an equation, leading to

$$x^1_{uv} + x^1_{vu} = z_k, \qquad uv \in C_k, \quad k \in \{0, \ldots, p\} \quad (7.9).$$

  And consequently, in Equation (7.19) of the formulation (W2) of page 111.

Contents

Chapter 1

Introduction

Various types of routing problems have been studied since the eighteenth century. Traditionally, these problems are stated in terms of graph problems in which some type of walk that satisfies certain conditions is sought. When looking for a walk that satisfies certain conditions, there are two different problems that one can address. The first one is the existence (or decision) problem of knowing if a route with the desired characteristics exists. Provided that such a walk exists, one can also face the optimization problem that consists of finding some optimal route among the ones that satisfy the required conditions.

Routing problems can be classified into two large classes depending on whether the conditions are stated on the nodes or the links of the considered graph. Problems of the first type are referred to as *node routing problems*, whereas problems of the second type are referred to as *arc routing problems*. Examples of node routing problems are all the ones in which vehicles provide service on the nodes of the graph, like it happens in pick-up/deliveries of goods. On the other hand, examples of arc routing problems are garbage collection, mail delivery, road sweeping, maintenance of electrical or telephone networks, etc., and, in general, all types of problems in which vehicles must provide service on the links of a graph. Various public transportation problems belong both to the first and the second types.

Routing problems can also be classified according to the type of graph where they are defined. In particular, we can consider *undirected* graphs, where the links are edges associated with unordered pairs of vertices so that they can be traversed in both directions, *directed* graphs, where the links are arcs associated with ordered pairs of vertices and they can be traversed only in one direction, *mixed* graphs, where the links can be both edges and arcs, and *windy graphs*, which are undirected graphs where the cost function associated with the edges is not symmetric, so that the cost of traversing one edge in one direction need not be the same than the cost of the traversal of the edge in the opposite direction.

The study of node routing problems goes back to 1856 with the works of Kirkman [75], who established sufficient conditions on a polyhedral graph for the existence of a cycle that traversed exactly once all the nodes of the graph, and of Hamilton [68], who studied various problems related to cycles in graphs. These studies are the precursors of the most relevant combinatorial problem nowadays, which is the so called Traveling Salesman Problem (TSP), that consists of finding a minimum length route that visits each node of a complete graph exactly once. In terms of optimization problems, node routing problems belong to class $\mathcal{NP}$-hard. This means that, unless $\mathcal{P} = \mathcal{NP}$, there can be no algorithm that guarantees obtaining optimal solutions in times polynomially bounded on the size of the instances. Due to the theoretical interest and the enormous range of applications of routing problems, there is a vast literature on related topics that have appeared in the last fifty years. There are several books fully focused on the TSP, like the classical one edited by Lawler, Lenstra, Rinnooy-Kan and Shmoys [80], or also the one by Gutin and Punnen [67], and more recently by Applegate et al. [3].

A generalization of the TSP are Traveling Salesman Problems with Profits, otherwise known as Profitable Tour Problems. In these problems there is not a given subset of vertices to be visited. Instead, there is a profit associated with each serviced vertex. Different versions of Profitable Tour Problems have been studied. A first review of 1989 of the related problems and works can be found in Balas [14]. More recently, in Balas [15], and the surveys of Feillet, Dejax and Gendreau [52].

There are also many other types of node routing problems where additional considerations come into play. These include capacity on the vehicles, time windows and precedence constraints. Thorough surveys on node routing problems can be found in the works by Laporte [76], and in the book edited by Toth and Vigo [104] dedicated to Vehicle Routing Problems.

There also exists a large literature on arc routing problems. Comprehensive surveys are the ones of Assad and Golden [13], and Eiselt, Gendreau and Laporte [49]. A more recent monograph is the book edited in 2000 by M. Dror [45]. The first reference of the mathematical study of an arc routing problem is from 1736. It can be found in the work of Euler [50], who stated necessary and sufficient conditions for the existence of a closed route that traversed exactly once all the links of a graph. These conditions were later extended in 1873 to non closed routes by Hierholzer [70]. The seminal work in arc routing optimization is that of Guan [65], a Chinese mathematician who in 1962 addressed the so called Chinese Postman Problem (CPP), that consists of finding a minimum length route that traverses each link of a graph at least once. As opposed to most routing problems this problem is polynomially solvable both on an undirected graph [48] and on a directed graph [48, 92].

The Rural Postman Problem (RPP) is the generalization of the Chinese Postman Problem that results when the arcs that must be traversed need no longer be all the links of the graph, but a subset of them, called *required* links. This problem was proposed in 1974 by Orloff [92] and is $\mathcal{NP}$-hard in all its

versions. It has been studied by numerous authors. Among the most relevant contributions for the undirected case we must cite the works by Christofides et al. [28] who proposed the LP based algorithm for the RPP, Corberán and Sanchis [40], who have studied its polyhedral structure, and Ghiani and Laporte [58], who characterized the edges that can be traversed two times in the optimal solutions and proposed a new model based on this characterization. The Windy Rural Postman Problem has been studied in Benavent et al. [23], where a heuristic is proposed and in Benavent et al. [22], where heuristics and lower bounds are considered. Other recent works on arc routing problems are those of Corberán, Mota and Sanchis [36] that compare two different formulations for arc routing problems, and Corberán, Plana and Sanchis [37] that introduce *zigzag* inequalities, which is a new class of inequalities for arc routing problems.

The most general routing problem is the so called General Routing Problem (GRP). In the GRP, demand can be located both at some vertices or links of the graph so there exist both required vertices and links. The GRP was first proposed by Orloff [92]. It has also been studied by Corberán and Sanchis [41], where its polyhedral structure is studied, and by Corberán, Mejía, and Sanchis [35]. The GRP has also been studied by Corberán, Letchford and Sanchis [33] where a cutting plane algorithm is proposed.

As with any other type of routing problems, the windy version is the most general version of the GRP. This problem has been studied in the doctoral thesis of Plana [99], and there are recent relevant works, whose results also apply to other families of arc routing problems, like those of Corberán, Plana, and Sanchis [38, 39]. There are also many other types of arc routing problems where capacity on the vehicles, time windows or other type of considerations must be taken into account.

Arc routing problems with profits can be seen as the arc routing counterpart of Traveling Salesman Problems with Profits. As opposed to classical arc routing problems, in arc routing problems with profits there is no specific link subset to be serviced, but there are profits associated with the links that are serviced (in addition to the costs of the links). That is, the set of required links is not given and it must be decided, together with the route that will serve the selected links, so as to maximize the net profit. To the best of our knowledge, the literature on arc routing problems with profits is scarce. We are aware of the work of Deitch and Ladany [44] where the problem is transformed into a node-routing one, and the work of Feillet, Dejax and Gendreau [51] where a limit, possibly greater than one, is given on the number of times the visit of an edge is beneficial.

Prize-collecting Arc Routing Problems (PARPs) is a family of arc routing problems with profits that has been proposed recently. While in general, in arc routing problems with profits, the profit of every serviced edge can be collected as many times as the link is serviced, in PARPs it is supposed that the profit of each serviced link is collected at most once, when the link is serviced, independently of the number of times that it is traversed.

There are several works on PARPs, all of which exploit the fact that only the first traversal of edges is beneficial. Aráoz, Fernández and Zoltán [11] introduced the PARPs and studied some properties that gave rise to a formulation in terms of a Integer Linear Programming (ILP) problem. The basic version of a PARP, which is called the Prize-collecting Rural Postman Problem is studied by Aráoz, Fernández and Meza [10] where an exact algorithm is proposed.

In this thesis we address the Clustered Prize-collecting Arc Routing Problem (CPARP) defined on an undirected graph. The CPARP is a PARP where, in addition, we consider the components (clusters) defined by the edges with demand, and for each cluster we require that either all its links are serviced or no link of the cluster is serviced. To the best of our knowledge, so far the CPARP has not been studied in the literature. Many results of this thesis have been presented in Conferences [8, 7, 5] and are summarized in a paper which has been conditionally accepted for publication [6].

The motivation for studying the CPARP comes not only from its theoretical interest but also from its potential applications. PARPs appear in the context of private companies looking to maximize operational profits, so that demand edges will not be serviced unless they yield a profit to the company, and each demand edge would be serviced at most once. In the case of garbage collection, recycling goods collection, or street cleaning, among others, while it is not acceptable that only a part of a given neighborhood is serviced, the whole neighborhood might not be profitable for the servicing company.

Therefore, potential applications of the CPARP take place in such contexts. Examples arise, for instance, in several cities of Minnesota (US) or in Buenos Aires (Argentina) where garbage collection companies bid to the municipality for parts of the city, so that, depending on the area, service is given by a different company. Another potential application is mail delivery in Germany where, recently, postal services have been privatized, so that companies have the possibility of servicing different districts. Yet, another potential application comes from the European Community directive for the collection of electric and electronic equipment (WEEE). According to this directive, each producer should be responsible for financing the management of the waste from his own products. It is thus natural that the collecting activities through the drop-off points be given to subcontractors which, in turn, bid for the collection of the different areas.

Another source of potential applications comes from classical arc routing problems, by interpreting the profit of each required edge as the priority for servicing it. Thus, the sum of the profits of the served components can be seen as the overall priority of the served components. In this setting we may allow not to serve some required edge, but we want to maximize the overall priority of the served components. Hence, the CPARP consists of finding a set of required components and an Eulerian circuit with maximum net overall priority.

To some extent, the problem that we address is related to the Clustered

Rural Postman Problem (CRPP) studied by Dror and Langevin [46] where the connected components defined by the arcs with demand must be completely serviced before servicing any other component. However, there are several differences between the CPARP and the CRPP. The first one is that CRPP is an arc routing problem, but not a PARP. The second one is that CRPP is stated on a directed graph, whereas we consider the CPARP on an undirected graph. Finally, in the CRPP of [46] it is required that all components are serviced, and that a component is completely serviced before servicing any other component. This third constraint is used then for making possible to transform the problem into a Generalized Travelling Salesman Problem. In the CPARP we release these requirements. On the one hand, we do not require to service all the components. On the other hand, we allow not servicing all the edges of a component consecutively if this results in a better solution.

As we will see, when we pose the additional requirement that either all the links of a cluster are serviced or no link of the cluster is serviced, we can prove some dominance conditions that otherwise do not hold. These properties allow some transformation of the original graph that results in a stronger formulation of the problem. This formulation has an exponential number of inequalities of two types: Inequalities that guarantee the connectivity of the traversal with the depot, and inequalities that guarantee the even degree of the nodes, which are a particular case of the so called *cocircuit* inequalities of Barahona and Grötschel [17].

Due to the exponential number of constraints it is important to address the separation problems for both types of inequalities. As we will see, in both cases we can solve exactly the separation problem. We address the exact separation of connectivity inequalities with an algorithm that is an adaptation to the CPARP of the exact algorithm of Belenguer and Benavent [18]. Although the cocircuit inequalities can be separated exactly in polynomial time, as can be seen in the works of Letchford [82] and Aráoz, Fernández and Meza [10], for speeding up the process we propose a heuristic separation that gives very good results. This enables us to propose an efficient LP based iterative scheme that starts with a small number of inequalities and at each iteration reinforces the current formulation with violated inequalities.

We also study heuristics for generating feasible solutions to the CPARP whose values can be compared with the upper bounds associated with the optimal LP solution. The heuristic that gives the better results is based on the heuristic of Fernández et al. [53] for the RPP. It is based on building a spanning tree on selected subsets of clusters, and then adding ad hoc edges so as to obtain an Eulerian graph.

For the sake of completeness, we propose an exact branch and cut algorithm to optimally solve the CPARP. At the root node we obtain an upper bound by solving the LP relaxation of the model and a lower bound with a heuristic which exploits the information of the optimal LP solution.

For analyzing the performance of all the proposed algorithms we have run a series of computational experiments with a set of benchmark instances. As we will see, the numerical results are very satisfactory. They indicate that in most cases (more than 75%) the instances can be optimally solved at the root node of the search tree. The remaining instances required, in general, very few nodes to explore the search tree. All but two instances of the 118 considered ones were optimally solved in less than two minutes of cpu time.

The thesis is structured as follows. Chapter 2 introduces some background concepts and gives an overview on different types of routing problems. In Chapter 3, we present the CPARP, study its properties and propose a formulation as an ILP. Chapter 4, addresses the separation problem for the two exponential families of inequalities, and presents the iterative LP solver algorithm for solving the LP relaxation of the CPARP and the exact branch and cut algorithm. In Chapter 5 we present the heuristics that we have considered, whereas in Chapter 6 we describe the computational experiments with the proposed algorithms and we analyze the obtained numerical results. In Chapter 7 we study the Windy CPARP (WCPARP). As with other arc routing problems the windy version of the CPARP generalizes all possible versions of the problem. We assume that the profit function on the demand edges is symmetric, but the cost function associated with the traversal of edges is asymmetric. Several properties that hold when the cost function on the edges is symmetric, no longer hold for the WCPARP. In Chapter 7 we propose two formulations for the WCPARP and we give some preliminary computational results obtained with one of them. Finally, the thesis ends in Chapter 8 with some conclusions and comments on future research.

We have also included an appendix where we present the *filògrafus*. The *filògrafus* is an user friendly application developed entirely by the author of this dissertation, that gives a visual and interactive interface for different types of optimization problems in graphs.

Chapter 2

Background

Basically, two main parts compose this chapter. Some notation for the problem and preliminaries on the theory of polyhedra are presented in the first section. Along the rest of the chapter the best known routing problems with one vehicle are presented. As a matter of fact, the family of such problems is wide enough to go beyond the scope of this dissertation. For this reason, we only present the ones that are usually taken as a reference in the literature, as well as those that in some way can be seen as precursors of the CPARP, which is the object of this thesis. We will state each problem on a graph, and we will recall some of their most relevant characteristics.

2.1 Notation and Preliminaries

Some notation and nomenclature is introduced in this section. We also give some basic concepts and describe some techniques that will be used later on.

An *undirected graph* G of *order* n and *size* m is denoted as $G = (V, E)$ where V is a set of *vertices* and E is a multiset set of *edges*, or unordered pairs of vertices of V, with $n = |V|$ and $m = |E|$. The undirected *complete* graph of n vertices, denoted K_n, is the graph whose edge set E has all possible pairs of the given n vertices. Instead, when an order is implicit on the edges, a *directed* graph (or *digraph*) D is denoted as $D = (N, A)$. In this case, its vertices are called *nodes*, N is the set of nodes, and the ordered pairs linking these nodes, the elements of A, are called *arcs*. A *mixed* graph has both kinds of *links*, arcs an edges, and is denoted by $G = (V, E \cup A)$.

Unless otherwise stated we assume that we work on an undirected graph. Thus no order is implicit on the edges $e = uv = vu \in E,\ u, v \in V$.

We may refer to the vertices of the graph G by $V(G)$ and to its edges by $E(G)$. We use the notation $\delta(S) = \{uv \in E|\ u \in S \text{ and } v \in V \setminus S\}$ for making reference to the *cut* of the subset of vertices $S \subset V$ and $S \neq \emptyset$. When $S = \{u\}$, $u \in V$, i. e., $|S| = 1$, then the cut of vertex u is $\delta(u) = \{uv \in E|\ v \in V\}$ and the *degree* of u is $|\delta(u)|$. From a graphical point of view, the cut of a subset are the edges in its frontier. For making reference to some edges of the cut of a set of vertices S we will also use the notation $\delta_B(S) = \delta(S) \cap B, \quad B \subseteq E$. Thus $\delta(u) = \delta_E(u)$. Also, for all subsets $S \subseteq V$, $|S| \geq 2$, we denote $\gamma(S) = \{e = uv \in E|u, v \in S\}$ the set of edges that link two vertices of S.

Finally, for a given a real function defined on the set of edges $f : E \to \mathbb{R}$ that associates a real number $f_e \in \mathbb{R}$ with each edge $e \in E$, we use the compact notation,

$$f(B) = \sum_{e \in B} f_e, \quad B \subseteq E.$$

A *walk* from vertex u to vertex v on a graph is an alternate sequence of vertices and edges starting in u and ending in v. The edges connect each pair of consecutive vertices in the sequence. A walk of *length* l is represented by $W(v_0, v_l) = v_0, e_1, v_1, e_2, v_2, \ldots, e_l, v_l$, where $e_i = (v_{i-1}, v_i)$. When $v_0 = v_n$ it is a *closed* walk. Throughout, the term *tour* is used for making reference to closed walks. When no edge is used more than once, the walk is said to be *simple*. A *path* is a simple walk. A *cycle* is a simple tour with no repetition of vertices, except for the first one.

A set $H \subset \mathbb{R}^n$ is called *halfspace* if there is a vector $a \in \mathbb{R}^n$ and a scalar a_0 such that $H = \{x \in \mathbb{R}^n \mid a^T x \leq a_0\}$. We say that H is the halfspace defined by the inequality $a^T x \leq a_0$, and we also say that if $(a \neq 0)$ the hyperplane $\{x \in \mathbb{R}^n \mid a^T x = a_0\}$ is the hyperplane defined by $a^T x \leq a_0$.

An inequality $a^T x \leq a_0$ is *valid* with respect to $S \subset \mathbb{R}^n$ if $S \subset \{x \in \mathbb{R}^n \mid a^T x \leq a_0\}$, i.e., is S is contained in the halfspace defined by $a^T x \leq a_0$. A valid inequality $a^T x \leq a_0$ for S is called *supporting* if $S \cap \{x \in \mathbb{R}^n \mid a^T x = a_0\} \neq \emptyset$.

A *polyhedron* is the intersection of finitely many halfspaces, i.e. every polyhedron P can be represented in the form $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$, being $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^n$. A bounded polyhedron (i.e. a polyhedron P such that $P \subset \{x \in \mathbb{R}^n \mid \parallel x \parallel \ \leq M\}$ for some $M > 0$ is called a *polytope*. Polytopes are sets $\in \mathbb{R}^n$ which are the convex hulls of finite number of points. Every polytope P can be written as $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$, and as $P = conv(S)$, $S \subset \mathbb{R}^n, |S|$ finite.

A subset F of a polyhedron P is called a *face* of P if there exists an inequality $a^T x \leq a_0$ valid with respect P such that $F = \{x \in P \mid a^T x = a_0\}$. We say that the inequality $a^T x \leq a_0$ defines F. A face F is called proper if $F \neq P$. A *facet* F of a polyhedron P is a proper, nonempty face (i.e. a face satisfying

$\emptyset \neq F \neq P$) which is maximal with respect to set inclusion.

The above definition of a polyhedron is used when analyzing a Linear Programming (LP) problem. This is, the set of feasible solutions of an LP problem $\max\{c^T x \mid Ax \leq b,\ x \geq 0\}$ is a polyhedron $P = \{x \in \mathbb{R}^n \mid Ax \leq b,\ x \geq 0\}$. In order to apply linear programming techniques, polyhedra have to be given in the form $\{x \in \mathbb{R}^n \mid Ax \leq b\}$. For this purpose valid inequalities and in particular, facets are therefore of particular importance.

Branch and cut is a technique used to solve ILP problems. It is a branch and bound method with some additional ingredients. Next we give an overview of this technique. Further details on the branch and cut are given in Padberg and Rinaldi [94], Jünger, Reinelt and Rinaldi [72], Jünger, Reinelt and Thienel [86] and Caprara and Fischetti [27].

The linear relaxation problem of an Integer Linear Programming (ILP) problem is the LP problem obtained from the ILP by dropping the condition that all variables have to be integers. Therefore, the optimal value z_{LP} of the relaxation (in the maximization case) is an upper bound to the optimal z_{ILP} of the ILP program, i.e., $z_{LP} \geq z_{IP}$.

If the number of constraints of an ILP problem is small enough, a classical solution method is via branch and bound using linear programming bounds. We suppose that the integer variables are indexed by a set E and e is an element of E (i.e. $e \in E$). First, we solve the linear relaxation, if the optimal solution x^* is integral we are done. Otherwise, we choose a variable x_e with fractional value x_e^* in the linear relaxation problem to build two new LP problems. In the first problem we add the constraint $x_e \geq \lceil x_e^* \rceil$ (where $\lceil x_e^* \rceil$ is the smallest integer greater than x_e^*) and in the second problem we add the constraint $x_e \leq \lfloor x_e^* \rfloor$ (where $\lfloor x_e^* \rfloor$ is the greatest integer smaller value than x_e^*). The branch and bound method continues building new LP problems in a tree structure with a *node* for each problem. A branch is cut off (fathomed) at a node either when its optimal solution is integer (improving the current optimal integer solution) or when its optimal solution value is lower or equal than the lower bound. When the number of linear constraints is large, that is, its size depends exponentially on some parameter that depends polynomially on the size of the problem, the constraint system cannot be given to an LP solver and a *cutting plane* technique has to be used to solve the linear program.

Let I be an ILP problem and L be the LP problem consisting of its relaxation, possibly strengthened with additional valid inequalities, having a very large number of constraints (that is, $\Omega(2^n)$). Let z_I and z_L be their optimal solution values. Let us assume that we want to maximize the objective function. The cutting plane works as follows. For an iteration i, $i > 0$, let L_i be an LP problem consisting of a subset of a reasonable number of constraints in L (that is, $\mathcal{O}(n^k)$ for some $k \in \mathbb{Z}_+$). Solve L_i, which yields an optimal solution x_i. If this solution is feasible for the ILP problem I, then it is optimal for I. Otherwise, we solve the separation problem that consists in either finding one constraint of L violated by

x_i, or proving that such constraint does not exist. If some violated constraints are returned, L_{i+1} is obtained by adding them to L_i. Then we solve L_{i+1} and repeat the procedure, otherwise terminate. Note, therefore, that for every $i > 0$, if z_i is the optimal value of L_i, we have $z_i \geq z_{i+1} \geq z_L \geq z_I$.

We usually end up with an optimal solution to the ILP problem I or to its linear relaxation L, but sometimes the solution to the separation problem is not exact. That is, it may do not return a violated constraint when there is at least one. In these cases, let us denote L^∞ the LP problem in which the cutting plane technique has terminated with a solution not optimal for I. Then, x^∞ is not integer. Thus, we can split the problem into two new problems by adding upper and lower bounds to a variable x_e^∞ whose current value is fractional, as is done in a branch and bound algorithm. This process is called *branching*. We proceed by recursively solving (with the cutting technique explained above) each new problem, and controlling the enumeration tree as a branch and bound algorithm. The method combining both techniques is called a *branch and cut*.

Enumeration and cutting plane techniques benefit from each other: on the one hand, the bound produced at each node of the enumeration tree is better than in a branch and bound algorithm, because new inequalities are added to the formulation of the corresponding subproblem. On the other hand, the separation algorithm takes advantage from the branching process as it produces a perturbation on the fractional solution that could not be cut away by an inequality.

2.2 Routing Problems

Routing Problems are optimization problems stated on graphs whose solutions consist of tours. Besides the graph, a real cost function on the edges is given. Usually, the tour must fulfill some problem specific requirements while minimizing the total cost of its edges.

In routing problems a variable is associated with each edge indicating the number of traversals of it in the solution tour. In this way a tour is represented as a point $x \in \mathbb{Z}^m$, since the number of traversals of each edge must be integer in order to the solution be feasible.

Given a routing problem on a graph G, we denote the convex hull of all feasible tours, or feasible solutions, by $P(G)$, that as can be proved is a polyhedron. When the variables associated with the edges that make part of the tours are binary, the polyhedron is a polytope since it is bounded. Instead, when the variables associated with the edges that make part of the tours represent the unbounded number of times that an edge is traversed, then the polyhedron of the problem is unbounded.

Routing problems may be classified according to several different criteria. Mainly, we distinguish among them depending on whether the requirements are stated on the nodes or on the edges of the graph. Hereby, we obtain the following classification:

- *Node Routing Problems* When the conditions that must be satisfied are established on the vertices.
- *Arc Routing Problems* When the conditions are stated on the links.
- *General Routing Problems*, When the requirements are stated both on the vertices and the links of the graph.

Alternatively, we may also classify routing problems depending on the type of graph on which the problem is stated. From this point of view we have:

- *Undirected problems* When the problem is stated on an undirected graph.
- *Directed problems* When the problem is stated on a directed graph.
- *Mixed problems* When the problem is stated on an mixed graph, i.e. when there are both kinds of links in the graph.

One more type of problem can be considered: *Windy problems*. Windy problems are stated on undirected graphs, but the cost of the edges depends on the direction of their traversal. The interest of these problems is based on the fact that windy problems constitute a generalization of all the others.

Other extensions not considered in this dissertation are problems with multiple vehicles with limited capacities, or time windows. In this thesis we work with just one vehicle. Problems with k vehicles have been studied by Frederickson [56], Pearn [96], and Ahr and Reinelt [1].

We also assume there are no capacity constraints nor time windows. Such extensions could be studied in future research. Problems with capacitated vehicles were first formulated by Christofides [28] and Golden and Wong [59]. These problems have been analyzed by Assad, Golden, and Pearn [12], and by Belenguer and Benavent [18] that proposed an exact branch and cut algorithm. Exact algorithms for capacitated problems have also been proposed by Welz [105], Hirabayashi, Saruwatari and Nishida [71], and Greistorfer [61].

Other problems are related to the scheduling of the service times. Problems with deadline classes have been solved with exact algorithms by Letchford and Eglese [84]. And also problems considering k vehicles with time windows have been treated by Clark and Wright [32].

With respect to the transformation of arcs routing problems into node routing problems, some works have been published among which we mention the work of Assad, Golden, and Pearn [97] related to the Capacitated Arc routing Problem or the work related to the Clustered Rural Postman Problem presented by Dror and Langevin [46]. As can be observed, the variety of routing problems is prolific.

Except for the Chinese Postman Problem (CPP) all the other problems described in this chapter are $\mathcal{NP}$-hard problems. This means that, unless $\mathcal{P} = \mathcal{NP}$, there is no algorithm that guarantees to obtain an optimal solution to the problem for any instance of size n in polynomially bounded time, that is a time in $\mathcal{O}(n^k)$ for some $k \in \mathbb{Z}_+$.

2.3 Node Routing Problems

Given that this dissertation does not focus on node routing problems, only four problems of this family are presented in this section. First, the Traveling Salesman Problem (TSP) and the Graphical Traveling Salesman Problem are studied, since these are possibly the two most emblematic routing problems. The third node routing problem that we study here is the Generalized Traveling Salesman Problem. The Generalized TSP can be seen as the node version of the Clustered Prize-collecting Arc Routing Problem, which is the object of this dissertation. In this sense, it might be considered as a precursor. We end the section with the Profitable Tour Problem, that is the TSP with profits, when both the cost function on the edges and the profit function on the vertices are present in the objective function. Due to its theoretical interest we will get into some detail for the first one, the TSP.

2.3.1 The Traveling Salesman Problem (TSP)

Definition 2.1 The Traveling Salesman Problem (TSP)
Given the complete graph of n vertices, K_n, and a positive real function of costs defined on its edges $c : E(K_n) \to \mathbb{R}_+$, the Traveling Salesman Problem is to find a cycle with the lowest cost, visiting each vertex exactly once.

Note that a solution for the TSP consists of a selection of n edges in the complete graph.

Therefore, the solutions of the TSP are cycles. Different formulations of this problem define a binary decision variable associated with each edge $e \in E$, to represent whether or not edge e is used in a solution. In this way, a cycle x can be represented by a vector of binary elements of dimension $m = |E|$.

$$x = (x_e | e \in E) \in \{0,1\}^m.$$

In order to establish a set of valid inequalities for the description of the polytope $TSP(G)$ we must consider properties that all solution cycles must fulfill:

$$x(\delta(u)) = 2, \quad u \in V \tag{2.1}$$
$$x(\gamma(S)) \leq |S| - 1, \quad S \subset V \tag{2.2}$$
$$x_e \in \{0,1\}, \quad e \in E \tag{2.3}$$

Observe that all tours satisfy that all their vertices have a degree of 2. This is what is imposed through the parity constraints (2.1), which also force the solutions to be composed by cycles. Another property that solution tours must satisfy is being connected. Inequalities (2.2) prevent solution tours from forming disconnected cycles. In fact, there are alternative ways to forbid the formation of disjoint cycles. However, this way is simple enough: Since in a cycle the number of edges is exactly the same than the number of vertices, constraints (2.1) together with constraints (2.2) prevent the cycles of length smaller than $n = |V|$.

The integrity constraints (2.3) must be posed since ignoring them may lead to fractional solutions, thus not feasible. In order to avoid these fractional values, we must identify inequalities that characterize the polytope associated with the problem. A great effort has been done, and still is being done nowadays, to achieve this, and to relate the polytope associated with the TSP to other polyhedra which are well characterized. Two of the simplest and best known ones are mentioned next.

For example, the relation between tours and *1-trees* is well-known. Let us recall that a 1-tree, *T1*, of G, consists of a spanning tree of G plus an additional edge. Since 1-trees are connected structures with n vertices and just one cycle, any TSP tour is also a 1-tree. On the other hand the polytope of all 1-trees of G, *T1(G)*, is fully described by a set of inequalities that are valid for the TSP. Thus, $TSP(G) \subset$ *T1(G)*.

Similarly, we can relate $TSP(G)$ with the *perfect 2-matching* polytope *M2(G)*. A perfect 2-matching for a graph G is a subset of edges such that each vertex in some edge of a *M2* is incident to exactly 2 edges of *M2*. Therefore, we can see that tours for the TSP are in turn perfect 2-matchings. So,

$$TSP(G) \subset (T1(G) \cap M2(G)).$$

With respect to the computational complexity, despite an intensive study by mathematicians, computer scientists, operations researchers, and others, for more than three decades, it remains an open question whether or not a general solution method exists to solve the TSP in polynomial time. In 1972, Karp proved that TSP is an $\mathcal{NP}$-hard problem [74].

It has been a major open problem for almost three decades to improve upon the best up to now approximation factor 3/2 given in 1976 by Christofides [29]. A large literature exists on this problem. Extensive monographs can be found in [3, 29, 63, 79].

2.3.2 The Graphical Traveling Salesman Problem

The Graphical Traveling Salesman Problem is a variant of the TSP introduced by Fleischmann [55] and Cornuèjols, Fonlupt and Naddef [43] in which the graph needs not be complete. In many applications this is more realistic than its predecessor since most often the underlying graph where the problem is stated is not a complete graph.

Definition 2.2 The Graphical Traveling Salesman Problem
Given a connected undirected graph $G = (V, E)$, and a positive real function of costs defined on its edges $c : E \to \mathbb{R}_+$, the Graphical Traveling Salesman Problem is to find a tour with the lowest cost visiting each vertex at least once.

Note that a solution for the Graphical TSP may duplicate some edges of G.

Hence, tours may be represented by

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

since we can not assume solutions to traverse just once each edge.

Now, x_e represents the number of times that the edge e is used in the tour x. All tours for the Graphical TSP can be represented as vectors x that satisfy

$$x_e \geq 0, \quad e \in E \tag{2.4}$$
$$x(\delta(u)) \equiv 0 \ (mod\ 2) \quad u \in V \tag{2.5}$$
$$x(\delta(S)) \geq 2 \quad S \subset V \tag{2.6}$$
$$x_e \in \mathbb{Z}$$

Inequalities (2.4) are called *trivial*, and do not need explanation. The *parity* inequalities (2.5) force vertices to be visited through an even number of traversals of edges. Observe, however, that in this form they are not linear. And the *connectivity* inequalities (2.6) impose that the cut of any subset of vertices must have at least two edges for being connected with the other vertices of the graph.

Let us point out that the convex hull of the feasible solutions of Constraints (2.4)-(2.6), that we call $Graphical\_TSP(G)$, is an unbounded poly-

hedron, since if an edge e is traversed in a feasible solution tour $\mathcal{T}$, any other tour obtained by adding to $\mathcal{T}$ an even number of traversals of e will also be feasible. Furthermore, the polytope of the TSP, $TSP(G)$, is a facet of the polyhedron $Graphical\_TSP(G)$. This is,

$$TSP(G) = Graphical\_TSP(G) \cap \{x \in \mathbb{R}^m | x(E) = n\}.$$

The complexity of the Graphical TSP is closely related to that of the TSP and therefore, the Graphical TSP is also an $\mathcal{NP}$-hard problem.

2.3.3 The Generalized Traveling Salesman Problem

The Generalized Traveling Salesman Problem assumes that the set of vertices of a given graph has been partitioned in a collection of sets. The objective for the Generalized TSP is to find a minimum cost tour which visits exactly one vertex of each set. When each set of vertices has cardinality 1 the Generalized TSP reduces to the classical TSP. The Generalized TSP has been studied extensively by Laporte, Mercure, and Norbert [77, 78], and Noon and Bean [88, 89]. A more recently work can be found in Fischetti, Salazar and Toth [54].

2.3.4 The Prize-collecting Traveling Salesman Problem

The Prize-collecting Traveling Salesman Problem (PTSP) is the TSP with profits on the vertices. Several approaches have been done to this problem.

In 1989, the PTSP was introduced by Balas [14, 15]. Nevertheless, the profit objective is stated as a constraint. The aim is to find a circuit that minimizes travel costs and whose collected profit is not smaller than a preset value.

Dell Amico, Maffioli and Värbrand [2] stated the PTSP, which is a generalization of the TSP, where it is not necessary to visit all vertices. The overall goal is the simultaneous optimization of the collected profit and the travel costs.

Recently, Feillet Dejax and Gendreau [52] modelled different approaches and compared exact and heuristic solution techniques.

2.4 Arc Routing Problems (ARPs)

Arc Routing Problems (ARPs) deal with finding out least cost traversals in graphs, taking into account some constraints imposed on some links of the graph. A good overview of the area of arc routing can be found in the work by Assad

and Golden [13], although from 1995. An extensive monograph on this type of problems can also be found in the book by Dror [45], making a compilation from different authors.

Problems like planning the school transportation routes, post mail delivery, garbage pickup, road network maintenance, or the irrigation in the agriculture can be modelled as arc routing problems, see the paper by Bodin and Kursh [25]. And there is an increasing interest of such models in the field of telecommunication networks. Nevertheless, more humble uses might also feed the theory of ARPs. Take for instance the pastime pages of any magazine, when the problem is to draw some figure without raising the pencil of the paper.

In the next sections we will describe the ARPs with just one vehicle. The definitions raise in a natural way depending on whether demand for service is in all links (edges or arcs) of a graph or just in some of them.

2.4.1 The Chinese Postman Problem (CPP)

Next, the most general definition of the CPP on a mixed graph is given.

Definition 2.3 The Chinese Postman Problem (CPP)
Given a strongly connected graph $G = (V, E \cup A)$ with a positive real function of costs defined on its links $c : (E \cup A) \rightarrow \mathbb{R}_+$, the Chinese Postman Problem is to find a closed walk with the lowest cost that traverses all links in $E \cup A$ at least once.

Note that a solution to the CPP may require duplicating some links.

Different cases are of interest: the Undirected CPP where $A = \emptyset$, the Directed CPP where $E = \emptyset$, the Mixed CPP where $A \neq \emptyset$ and $E \neq \emptyset$ and the Windy Postman Problem where the costs for traversing an edge $e = uv = vu$ depend on the direction in which is traversed, so that $c_{uv} \neq c_{vu}$.

The Undirected CPP (UCPP)

The decisional version of the UCPP is the Euler [50] problem stated in 1736 on the Königsberg bridges. Broadly speaking this problem consists of finding a tour that traverses all the links of a graph exactly once, this is, a simple tour. It is well known that such a walk exists if and only if all the nodes of the graph have even degree. Eulerian graphs are those where all vertices have even degree. We also owe to Leonard Euler, the theorem stating that the number of odd vertices in any graph is even.

The idea of *minimizing* the length of a walk passing through each edge of a graph, stating hereby an optimization problem, was originally stated in 1962 by Kwan Mei-Ko [65], also known as Guan. He proposed an algorithm that started from the idea of pairing the odd-degree vertices of the graph thus making it Eulerian. With this algorithm he introduced what is the core of arc routing: The augmentation problem, i.e.. the problem of determining the least cost way of making a graph Eulerian by introducing extra edges or arcs.

Definition 2.4 The Augmentation Problem
Given an undirected connected graph $G = (V, E)$ with $n = |V|$, and a positive real function of costs defined on its edges $c : E \to \mathbb{R}_+$, the Augmentation Problem is to find a minimum cost set of edges $M \subset E(K_n)$ such that $G(V, E \cup M)$ is Eulerian.

In 1973, this problem was analyzed by Edmonds and Johnson [48] who formulated the augmentation problem on an undirected graph using binary integer variables x_{uv}, $u < v$, representing the number of copies of edge uv that are added to the graph for making it Eulerian. Hence, the problem is formulated as

$$\begin{aligned} & & Minimize \quad & \sum_{e \in E} c_e x_e & \\ Subject\ to \quad & x(\delta(S)) \geq 1, \quad & S \subset V,\ |S_{odd}| \quad odd, & & (2.7)\\ & & x_e \geq 0, \quad & e \in E, & (2.8)\\ & & x_e \in \mathbb{Z}, \quad & e \in E. & (2.9) \end{aligned}$$

where for any $S \subset V$, S_{odd} denotes the subset of its odd-vertices.

In [48] was also proved that the polyhedron of solutions to (2.7) and (2.8) is the convex hull of feasible solutions to the UCPP.

Therefore, using the algorithm of Edmonds and Johnson [48] the UCPP can be solved in an efficient manner by first computing all shortest paths between all odd-degree vertices and then solving a perfect matching problem over the set of these odd-degree vertices using these costs. Christofides [28] worked on the CPP and introduced the Capacitated CPP. Analysis of heuristics for the CPP have been published by Benavent et al. [21].

In fact, the most remarkable feature that UCPP presents is that it is possible to solve it in polynomial time with respect the size of the instance. The complexity of the problem is dominated by the resolution of the perfect matching problem, that is solvable in $\mathcal{O}(n^3)$.

The UCPP, and the DCPP, are the only ARPs for which we can give exact algorithms bounded by polynomial time.

The Directed CPP (DCPP)

In the Directed CPP the problem is stated on a directed graph, so arcs must be traversed in the corresponding direction. Then, we use the notation $|\delta^-(v)|$ and $|\delta^+(v)|$ for the in- and out-degree of the vertex v.

The condition under which a strongly connected graph is Eulerian is that $|\delta^-(v)| = |\delta^+(v)|\,,\ v \in V$.

A procedure for solving the DCPP was proposed by different authors almost simultaneously in 1974 by Edmonds and Johnson [48], Orloff [92], and Beltrami and Bodin [19]. It is based on solving an auxiliary transportation problem.

To illustrate the idea, let us first denote by $s_u = |\delta^+(u)| - |\delta^-(u)|$, $u \in V$, and consider $V_s \subset V$ the subset of V with $s_u > 0$ of vertices acting as suppliers. Similarly, we denote $d_v = |\delta^-(v)| - |\delta^+(v)|$, $v \in V$, and define $V_d \subset V$ as the subset of vertices such that $d_v > 0$. Then, the problem can be formulated as

$$\begin{aligned}
Minimize \quad & \sum_{u \in V_s} \sum_{v \in V_d} c_{uv} x_{uv} \\
Subject\ to \quad & \sum_{v \in V_d} x_{uv} = s_u, \quad u \in V_s, && (2.10)\\
& \sum_{u \in V_s} x_{uv} = d_v, \quad v \in V_d, && (2.11)\\
& x_{uv} \geq 0, \quad u \in V_s,\ v \in V_d. && (2.12)
\end{aligned}$$

Inequalities (2.10) and (2.11) impose that for any supply vertex, the set of its outgoing arcs that are used in the solution must coincide with its surplus. Similarly, for any demand vertex, enough entering arcs must be used in any feasible solution.

The polytope of the DCPP is fully described by equations (2.10)-(2.12). Therefore, the DCPP is solvable in polynomial time as well.

Observe that in the formulation, integrality conditions on the variables are not required given that the matrix of the coefficients is totally unimodular (this is a matrix such that all its squared sub matrixes have determinant 0, 1, or -1) and, moreover, the quantities s_u and d_u are integer.

The Mixed CPP (MCPP)

The conditions for a mixed graph be Eulerian are

- The sum of in- and out-degrees must be even.
- The degree corresponding to undirected edges must also be even.
- For any subset $S \subset V$, $|\delta^+(S)| + |\delta^-(S)| < |\delta(S)|/2$.

In 1976, Papadimitriou [95] proved that the MCPP is $\mathcal{NP}$-hard by transforming it from 3SAT. In other words, if the MCPP could be solved in a time smaller than a power of the instance size n, then the problem known as 3SAT could also be, but it is well known that 3SAT is an $\mathcal{NP}$-hard problem and, unless $\mathcal{P} = \mathcal{NP}$, it can not. The problem remains $\mathcal{NP}$-hard even when assuming some simplifications on it, as requiring the graph being planar or having all costs equal.

Formulations define variables similarly than in the previous sections, in the sense that a variable associated with each link is defined. This problem has been studied by many authors. Exact algorithms have been given by Christofides et al. [20] using branch-and-bound technique with Lagrangean relaxation, Grötschel and Win [64], and Norbert and Picard [90] used branch-and-cut techniques. Laporte [76] proposed an exact algorithm by means of a transformation to the TSP. Also, Eiselt, Gendreau and Laporte [49] used network flow techniques to make the resulting Eulerian mixed graph completely directed.

On the other hand, an heuristic method has also been proposed by Edmonds and Johnson [48], later improved by Frederickson [57] who achieved a ratio of 5/3. Pearn and Liu [98] improved these existing heuristics. Raghavachari and Veerasamy [100] proposed different approximation algorithms improving the ratio to 3/2.

The Windy CPP (WPP)

This problem was first introduced in 1979 by Minieka [87]. The objective takes into account that not always the cost of traversing an edge must be equal in both directions.

Definition 2.5 The Windy Chinese Postman Problem (WPP)
Given an undirected connected graph $G = (V, E)$ with two positives real functions on the edges, that represent the costs of traversing an edge $uv \in E$, $u, v \in V$, in the direction from u to v and from v to u, the Windy Chinese Postman Problem is to find a directed tour with the lowest cost that traverses all edges of E.

The WPP has also been shown to be $\mathcal{NP}$-hard, by Brucker [26] and Guan [66]. Nevertheless, under certain conditions it might be solved in polynomial time.

The polytope of the WPP has been studied, and branch and cut algorithms have been proposed by Win [106, 107] and by Grötschel and Win [64].

2.4.2 The Rural Postman Problem (RPP)

The RPP was first introduced in 1974 by Orloff [92].

Definition 2.6 The Rural Postman Problem
Given a strongly connected graph $G = (V, E \cup A)$ with a positive real function of costs associated with its links $c : (E \cup A) \rightarrow \mathbb{R}_+$, and a subset of required links $R \subseteq (E \cup A)$, the Rural Postman Problem is to find a minimum cost tour that traverses all required links at least once.

Note that a solution to the RPP may select and duplicate some links.

When the set of required edges is connected the RPP reduces to a CPP. However, in the general case the RPP is in the class of $\mathcal{NP}$-hard problems in all its versions, as seen below. Similarly to the CPP, the RPP may be defined over undirected, directed, mixed or windy graphs. Many other variants of the RPP have been studied. Golden and Wong [59] introduced the capacitated version. Also, Letchford and Eglese [84] studied the RPP with deadline classes. Recently the Windy General Routing Problem has been studied in depth by Plana [99], and by Corberán, Plana and Sanchis [37], making a description of several families of valid inequalities most of which are facets, for the LP polyhedron of the WGRP. An exact algorithm for the WGRP was proposed by the same authors in [38]. The WGRP is a generalization of most ARPs, among which the windy version of the RPP is clearly included.

The Undirected RPP (URPP)

Many authors have studied the RPP when it is stated on an undirected graph. In 1981, Christofides et al. [30] proposed a branch and bound algorithm in which the lower bounds were computed by means of Lagrangean relaxation. The decision variables represented the number of copies of each edge to be added to the graph for making it Eulerian. Sanchis [102], and Corberán and Sanchis [40] proposed a different formulation. They also developed a branch and cut algorithm conducted by their polyhedral analysis. First, the original graph G_0 is transformed into a new graph G where the set of nodes is $V(R)$. That is, all the nodes that have no required edge incident with them have been eliminated. In addition, in G non required edges of G_0 have been substituted by

minimum cost paths between their end nodes. Like in the previous case, [30], decision variables represent the number of copies to be added to the set R in a feasible solution.

Let $V_i, i = 1, \ldots, p$ denote the set of vertices of the connected components induced by the set of required edges in G. Then, the basic ILP formulation of Corberán and Sanchis [40] is,

$$
\begin{aligned}
& & Minimize \quad & \sum_{e \in E} c_e x_e \\
Subject\ to \quad & x(\delta(v)) \equiv 0 \ (mod\ 2) \quad & v \in V,\ \delta_R(v)\ even, \\
& x(\delta(v)) \equiv 1 \ (mod\ 2) \quad & v \in V,\ \delta_R(v)\ odd, \\
& x(\delta(S)) \geq 2, \qquad & S \subset \cup_{i \in Q} V_i,\ Q \subset \{1, \ldots, p\}, \\
& & x_e \geq 0\ and\ x_e \in \mathbb{Z} \quad & e \in E.
\end{aligned}
$$

Corberán and Sanchis [40] proposed several families of new valid inequalities for the URPP: K-C inequalities, Path-Bridge and Honeycomb. They proposed a solution algorithm based on the above formulation where valid inequalities were separated and iteratively incorporated to the formulation of the problem. Letchford [82, 83] proposed new valid inequalities for the URPP. Ghiani and Laporte [58] proposed a new formulation based on the characterization of the variables that can take the value two in an optimal solution, and proposed an exact algorithm based on their formulation. Also, Fernández et al. [53] proposed a new formulation and obtained tight lower bounds for the URPP.

On the other hand, a heuristic for the URPP was proposed by Frederickson [57], based on the heuristic for the TSP given in 1976 by Christofides [29]. The cost matrix must satisfy the triangular inequality to guarantee the worst case ratio of 3/2.

With respect to the computational complexity Lenstra and Rinnooy Kan [81], proved that URPP is $\mathcal{NP}$-hard by transformation from HAMILTONIAN CIRCUIT.

The Directed RPP (DRPP)

The directed version for the RPP has also been studied in depth. Christofides et al. [31] gave an ILP formulation and proposed a branch and cut solution algorithm based on that formulation, where Lagrangean relaxation was used for obtaining lower bounds.

On the other hand, a similar heuristic to that of the undirected case consists in computing the minimum spanning tree. Analogously to the DCPP, once

the tree has been computed a transportation problem must be solved. This algorithm was also proposed by Christofides et al. [31]. One more heuristic algorithm was presented by Ball and Magazine [16].

The Mixed RPP (MRPP)

Given that the mixed version of the RPP is close to the windy version, the MRPP has been scarcely studied. A complete analysis was presented by Romero [101]. More recently, a heuristic procedure based on a tabu search algorithm for the MRPP has also been published by Corberán, Marti and Romero [34].

2.4.3 The Windy General Routing Problem (WGRP)

In the WGRP some vertices, edges, or both might be required. This problem was studied by Plana [99], and by Corberán, Plana and Sanchis [37]. This is the most general version of the RPP so far. Transforming the WGRP to any of the previous versions is simple.

Corberán, Plana and Sanchis [38] developed a polyhedral analysis of the WGRP identifying a large family of facets and introducing other valid inequalities.

We describe the formulation proposed by Plana [99], and Corberán, Plana and Sanchis [37]. The decision variables x_{uv} represent the number of times that edge $e = uv$ is traversed in the direction from vertex u to vertex v.

$$
\begin{aligned}
Minimize \quad & \sum_{uv \in E} (c_{uv} x_{uv} + c_{vu} x_{vu}) & \\
Subject\ to \quad & x_{uv} + x_{vu} \geq 1 \quad uv \in E_r & (2.13)\\
& \sum_{uv \in \delta(u)} (x_{uv} - x_{vu}) = 0 \quad u \in V & (2.14)\\
x(\delta(S)) \geq 1 \qquad & S \subset \cup_{i \in Q} V_i,\ Q \subset \{1, \ldots, p\} & (2.15)\\
& x_{uv}, x_{vu} \geq 0 & (2.16)\\
& x_{uv}, x_{vu} \in \mathbb{Z} & (2.17)
\end{aligned}
$$

The objective function is the sum of costs of the edges that are used taking into account the direction in which they are traversed. The *obligation* inequalities (2.13) force solutions to traverse each required edge, no matter in which direction. Then, the *flow conservation* (2.14) equations guarantee the parity of the vertices by imposing that the number of times that the solution tour enters

a vertex be equal to the number of times it leaves that vertex. Finally, the connectivity inequalities (2.15) make the solution connected.

2.4.4 The Clustered RPP (CRPP)

The Clustered Rural Postman Problem (CRPP) is a restricted version of the DRPP in which each connected component of arcs has to be completely serviced before servicing another component. To some extent, the problem that we address in this dissertation is related to the CRPP studied by Dror and Langevin [46] where the authors presented an enumerative solution method based on transforming the CRPP into a version of the Generalized TSP seen in subsection 2.3.3. Hence, the interest of this approach relies on this transformation more than in any LP formulation.

2.5 ARPs with Profits

In all the ARPs that we have presented so far, we have assumed that there is a set of mandatory or required links with demand that any feasible solution must service and, thus, the objective function only considers the cost of the additional links that are traversed in the routes. Potential applications of these problems arise naturally in the context of essential services. However, it is natural to consider other types of applications associated with non essential services, where there is a set of links with demand, but it is not mandatory to serve all such links. Now, if a demand link is serviced, a profit is obtained. As usual, each time that a link is traversed a cost is incurred. Therefore, in ARPs with profits the set of demand edges to be serviced must be decided. In addition, the minimum cost Eulerian tour that traverses all serviced links must be found.

To the best of our knowledge, the literature on arc routing problems with profits on the edges is very scarce. Apart from the previous works of Aráoz, Fernández and Zoltán [11], and Aráoz, Fernández and Meza [10], we are aware of only two works, one by Deitch and Ladany [44], and another one by Feillet, Dejax and Gendreau [51], where problems of this type are considered. However, in both papers the setting and the approach are quite different from that of [11, 10]. While the approach of [11, 10] address the problem as an arc-routing one and exploits explicitly the fact that only one traversal of each edge is beneficial, in [44] the problem is transformed into a node-routing one, whereas in [51] a limit, possibly greater than one, is given on the number of times the traversal of an edge can be beneficial.

2.5.1 The Prize-collecting RPP (PRPP)

An extension of the RPP was proposed in 2003 by Aráoz, Fernández and Zoltán [11]. It is called the Prize-collecting Rural Postman Problem (PRPP). Formally, the PRPP is defined next.

Definition 2.7 The Prize-collecting Rural Postman Problem
Given an undirected connected graph $G(V, E)$ with a distinguished vertex d (the depot), a real positive function of costs defined on its edges $c : E \to \mathbb{R}$ and a non-negative real function of profits, $b : E \to \mathbb{R}$ the Prize-collecting Rural Postman Problem is to find a closed walk, $\mathcal{T}^*$, which maximizes the value of

$$\sum_{e \in \mathcal{T}} (b_e - c_e x_e)$$

where $\mathcal{T}$ is any tour in G passing through d, and x_e is the number of times that the edge e is traversed in $\mathcal{T}$.

Thus, in the PRPP a profit is assigned to some edges, called edges *with demand*. This profit may be understood as a consequence of the service offered. So, there are not required edges but demand ones, and as opposite to the RPP, the PRPP admits the possibility of not servicing some edges with demand in the feasible solutions. In fact, the empty solution, consisting in not moving from the depot, is feasible for the PRPP as opposed to all problems presented so far.

A deeper analysis might be found in [11], and an exact algorithm for the problem is given in [10]. Note that the profit of an edge is collected only when it is first traversed by the solution tour. In [10], the authors proved that there is an optimal solution where no edge is traversed more than two times. This allows the authors to give a formulation that uses two sets of binary variables:

For each edge $e \in E$, we have

- x_e with value of 1 if edge e is traversed by the solution, and
- y_e, with value 1 when edge e is traversed twice in the solution.

Thus $x_e, y_e \in \{0, 1\},\ e \in E$. Given the relationship between the PRPP and the problem studied in this dissertation, the ILP formulation for the PRPP of [10] is presented here.

Attention should be paid to the fact that now a maximization objective function is considered, where some of the coefficients are given by $\varphi_e = b_e - c_e$. This objective function is divided in two terms associated with the first and the second traversal of the edges.

$$Maximize \quad \sum_{e \in E} \varphi_e x_e - \sum_{e \in E} c_e y_e$$

$$x(\delta(S) \setminus F) + y(F \setminus L) \geq x(F) + y(L) - (|F| + |L|) + 1, \quad (2.18)$$
$$S \subset V \setminus \{d\}, F \subseteq \delta(S), L \subseteq F, |F| + |L| \ odd$$

$$x(\delta(S)) + y(\delta(S)) \geq 2x_e, \quad (2.19)$$
$$S \subseteq V \setminus \{d\}, e \in \gamma(S), V(e) \cap V_R = \emptyset$$

$$x(\delta(S)) + y(\delta(S)) \geq 2x_{e_k^R}, \quad (2.20)$$
$$S \subseteq V \setminus \{d\}, k \neq 0, V_k \cap S \neq \emptyset$$

$$x_e = x_{e_k^R}, e \in \gamma(V_k) \cap R, k \in P \quad (2.21)$$
$$y_e \leq x_e, e \in E \quad (2.22)$$
$$x_e \leq x_{e_k^R}, e \in (\gamma(V_k) \cup \delta(V_k)) \setminus R, k \in P \quad (2.23)$$
$$y_e = 0, e \in \gamma(V_k) \setminus R, k \in P \quad (2.24)$$
$$x_e = 1, e \in \gamma(V_0) \cap R$$
$$x_e, y_e \in \{0, 1\}, e \in E$$

- Inequalities (2.18) ensure even degree at every subset of vertices and, in particular, at every vertex. We call *set*-parity inequalities to constraints (2.18), and *node*-parity inequalities to the particular cases with S being a singleton, i.e. $S = \{v\}$, $v \in V$. Inequalities (2.18) are an adaptation to the PRPP of the so called *cocircuit* inequalities introduced by Barahona and Grötschel [17] in a matroid context (see also Aráoz et al. [4]). These inequalities are otherwise known as 2-matching or parity inequalities. Broadly speaking they require that if a solution uses an odd number of edges incident with a subset of vertices S, then the solution uses at least one additional edge of the cut-set $\delta(S)$. In the case of the PRPP [10] the authors further exploit the precedence relationship of x variables with respect to y variables given by a dominance relation also proved by the authors, that allows to further restrict the set L to be a subset of F. The particular case of set-parity inequalities when F is a singleton, i.e. $F = \{e\}$ and $L = \emptyset$, also imply connectivity with the depot of any edge $e \in \delta(S)$ that is selected (that is, such that $x_e = 1$). Indeed, when $F = \{e\}$, and $L = \emptyset$, inequalities (2.18) can be rewritten as

$$\begin{aligned} x(\delta(S) \setminus F) + y(F \setminus L) \geq x_e + 1 - 1 \equiv \\ x(\delta(S)) - x_e + y_e \geq x_e \equiv \\ x(\delta(S)) + y_e \geq 2x_e \end{aligned} \quad (2.25)$$

  We denote this particular case as parity-connectivity inequalities.

- In (2.19) and (2.20), $R = \{e \in E \mid b_e > 2c_e\}$ and e_k^R is an arbitrarily selected demand edge of the kth. connected component induced by the set

R. Then, inequalities (2.19) and (2.20) ensure connectivity with the depot of any edge $e \in \gamma(S)$ that is traversed (that is, such that $x_e = 1$). These inequalities are referred to as connectivity inequalities. Inequalities (2.19) and (2.20), together with inequalities (2.18), guarantee that solutions to the model define Eulerian circuits.

- In [10] it is proved that whenever a demand edge e is serviced, all edges in $\delta_R(e)$ will also be serviced. This is imposed by equations (2.21).
- Inequalities (2.22) say that no edge can be traversed for the second time if it is not traversed for a first time.
- Equations (2.23) impose that if a non-demand edge e adjacent to some demand edge e_R is traversed, then e_R must be traversed too, and so, serviced.
- Inequalities (2.24) limit the edges that can be traversed two times in an optimal solution, by imposing that none of the non-demand edges in $\gamma(C_k) \setminus C_k$ can be traversed twice.

Chapter 3

The Clustered Prize-collecting Arc Routing Problem

In this chapter we study the Clustered Prize-collecting Arc Routing Problem, CPARP, defined on an undirected graph. Like in the PRPP seen in Subsection 2.5.1, there is a set of edges with demand, each of which produces a profit if it is serviced. We name *cluster* to each connected component defined by the set of demand edges. In the PRPP no condition was required on the set of demand edges serviced by feasible solutions. Instead, in the CPARP we impose additional constraints expressed in terms of conditions on the clusters. In particular, we require that in a feasible solution each cluster is either fully serviced (i.e. all of its demand edges are serviced) or not serviced at all (no profit of the edges of the cluster will be collected). We are not aware of any work that considers the problem addressed in this dissertation.

This chapter is structured as follows. We start by defining the problem. This includes a formal definition for the CPARP, an example presented next, and a reduction from the RPP that allows us to establish its complexity. Once the problem is presented, we get into the analysis of the model. Some properties of the problem can be postulated on the original graph. However, other properties only hold when the problem is stated on a complete graph. All these properties allow us to reduce considerably the set of edges that would make part of any optimal solution. Hence, we describe the transformation of the original problem into a problem defined on K_n being n the number of demand vertices, $n = |V(D) \cup \{d\}|$. Next, we prove a list of properties of feasible and optimal solutions for the CPARP when stated on the complete graph. Then, these properties are used for defining the sets of variables. We finish the chapter with an ILP formulation for the CPARP stated on the complete graph.

3.1 Definition

The CPARP is stated on an undirected connected graph $G(V, E)$. Before getting into the definition itself, we describe all input data that characterize an instance. Let $D \subset E$ denote the subset of demand edges, and $d \in V$ a distinguished vertex called the *depot*. Each cluster defined by the demand edges is denoted C_k, $k \in \{0, \ldots, p\}$, and the set of vertices of each cluster is denoted V_k. That is, $C_k = \gamma_D(V_k) = \gamma(V_k) \cap D$. For a simpler notation, we assume that $d \in V_0$. Throughout, we will use the terms D-even and D-odd to classify vertices depending on the parity of the number of demand edges in their cut. That is, $v \in V$ is D-even if $|\delta_D(v)|$ is even. Similarly, $v \in V$ is D-odd if $|\delta_D(v)|$ is odd. Let also denote $b : D \to \mathbb{R}_+$ a real positive function of profits on the elements of D, and $c : E \to \mathbb{R}_+$ a real positive function of costs on all edges of E.

Formally, the definition of the problem is as follows:

Definition 3.1 The Clustered Prize-collecting Arc Routing Problem (CPARP) *Feasible solutions for the CPARP are tours going through d, such that for each cluster C_k, $k \in \{0, \ldots, p\}$, either all its edges are serviced or none of its edges is serviced.*

The Clustered Prize-collecting Arc Routing Problem is to find a set of clusters $\mathcal{K}^* \subseteq \{0, \ldots, p\}$, and a tour $\mathcal{T}^*$, passing through d, that services all the edges in $\cup_{k \in \mathcal{K}^*} C_k$, but none of the edges in $D \setminus \cup_{k \in \mathcal{K}^*} C_k$, which maximizes the value of

$$\sum_{k \in \mathcal{K}} F_k - \sum_{e \in \mathcal{T}} t_e c_e$$

over all feasible tours $\mathcal{T}$, where $\mathcal{K}$ is the set of clusters serviced in the tour $\mathcal{T}$, t_e is the number of times that edge e is traversed in $\mathcal{T}$, and $F_k = b(C_k)$.

We represent a CPARP by *CPARP(G,D,d,b,c)*.

In the next subsection an example is presented. The same instance is used for the PRPP of Subsection 2.5.1 and for the CPARP in order to compare their optimal solutions. We denote by z^*_{PRPP} and z^*_{CPARP} the value of the optimal solutions for the PRPP and the CPARP, respectively. Note that, since the CPARP is a particular case of the PRPP, we will always have $z^*_{PRPP} \geq z^*_{CPARP}$.

Example

An example is depicted in Figure 3.1. Demand edges appear in bold, non-demand edges in thin lines. Next to each edge, its cost (in dark) and profit (in light) are depicted. In this example $V_0 = \{d\}$, $V_1 = \{1, 2, 3\}$ and $V_2 = \{4, 5, 6, 7\}$.

Also, $C_0 = \{d\}$, $C_1 = \{\{1,2\},\{1,3\},\{2,3\}\}$ and $C_2 = \{\{4,5\},\{5,6\},\{5,7\}\}$. Observe that edges $\{4,7\}$ and $\{6,7\}$ are in $\gamma(V_2) \setminus C_2$.

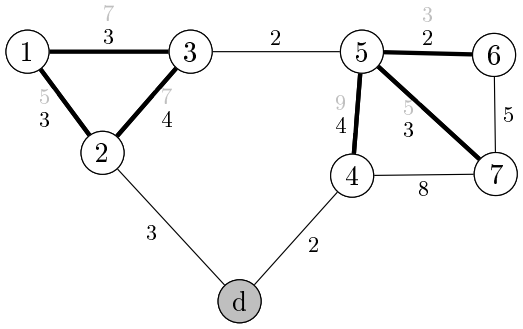

Figure 3.1: *Example instance. Costs are shown for all edges. Demand edges are in bold with its profits in light over their costs in dark.*

The unique optimal solution to the PRPP is the cycle $d-2-1-3-5-4-d$ with value $z^*_{PRPP} = 4$, that is $(5+7+9)-(3+3+3+2+4+2)$ as shown in Figure 3.2.

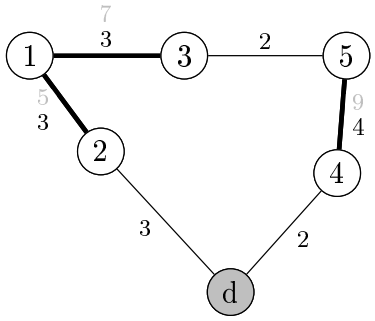

Figure 3.2: *Optimal solution to the PRPP instance of Figure 3.1, with* $z^*_{PRPP} = 4$.

However, for the CPARP this solution is not even feasible since in component C_1 edges $\{1,2\}$ and $\{1,3\}$ are serviced, but edge $\{2,3\}$ is not, and in component C_2 edge $\{4,5\}$ is serviced but edges $\{5,6\}$ and $\{5,7\}$ are not.

In Figure 3.3, the subgraph of demand edges D, composed by the two connected components plus the depot, is explicitly shown.

The optimal solution to this CPARP consists of servicing only the edges of component C_1 and is thus given by $d-2-3-1-2-d$ with value $z^*_{CPARP} = 3$, as depicted in Figure 3.4.

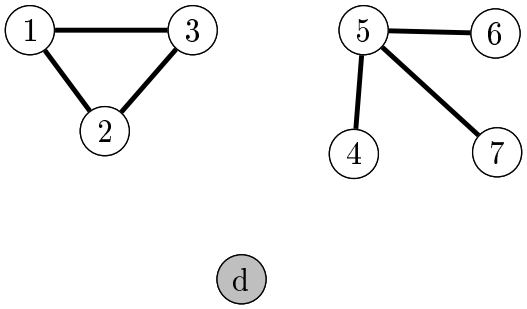

Figure 3.3: *Clusters of the instance of Figure 3.1.*

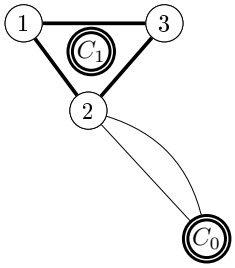

Figure 3.4: *Optimal solution to the CPARP instance of Figure 3.1, servicing clusters C_0 and C_1, with value $z^*_{CPARP} = 3$.*

Reduction from the Rural Postman Problem

The Rural Postman Problem is a particular case of the CPARP. Given a RPP on a graph G with cost function c and required set of edges R, let C_k, $k \in \{1, \ldots, p+1\}$, be the connected components associated with R, and let M be a very large constant, $M > 2c(E)$. Then, define the CPARP with demand set $D = R$, d any vertex in $V(D)$, set of clusters C_k, $k \in \{0, \ldots, p\}$, and b given by $b_e = M$, $e \in D$. If, furthermore, we denote w^k the cost of any bipath between d and C_k, we have

$$\sum_{e \in C_k} (b_e - 2c_e) = |C_k|\, M - 2c(C_k) > w^k \qquad k \in \{0, \ldots, p\}.$$

Therefore, the complete demand set $D = R$ will be serviced in any optimal solution to the CPARP, which defines an optimal solution to the initial RPP.

As a consequence of the above remark the CPARP is $\mathcal{NP}$-hard.

3.2 Analysis of the Problem

Most properties of the solutions to the CPARP are inherited from properties of the PRPP. Nevertheless, there are other properties that do not hold for the CPARP stated on a general graph. As we will see these properties certainly hold when the CPARP is stated on a complete graph. In particular, in [10] is proved that for the PRPP all optimal solutions that traverse a demand edge gives service to it. This property does not hold on the CPARP as defined in Section 3.1. This is illustrated in the example of Figure 3.5.

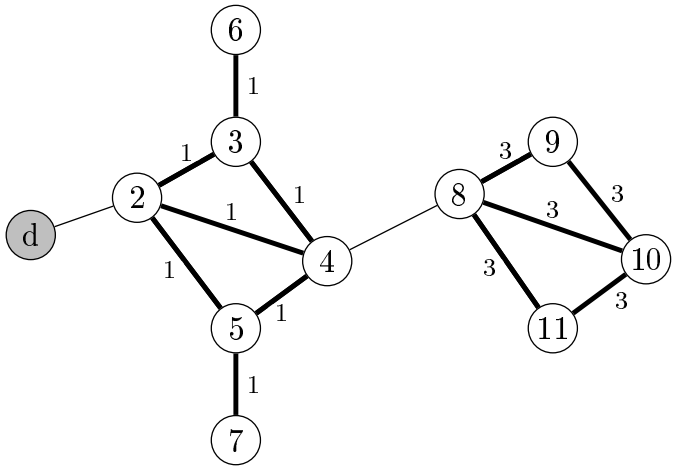

Figure 3.5: *An instance of a CPARP with three clusters. Costs are equal to 1 for all edges. Benefits are shown.*

In the instance of Figure 3.5, edges with demand define two components. In the first one the profit for servicing each edge is one, whereas in the second one it is three. All costs of the edges of the graph are one.

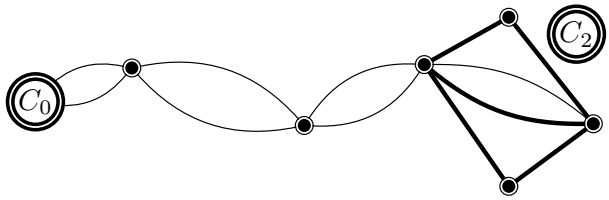

Figure 3.6: *Optimal solution to the CPARP instance of Figure 3.5, servicing clusters C_0 and C_2, with value $z^*_{CPARP} = 3$.*

It is easy to check that the optimal solution consists of servicing the second

component and not servicing the first one, thus resulting $z^*_{CPARP} = 3$. This optimal solution is depicted in Figure 3.6 with labels for the clusters serviced.

Observe that the optimal route does traverse, two times, the edge $\{2, 4\}$ of the first component. Hence, this example illustrates that there might exist optimal solutions to the CPARP where some edges with strictly positive profit are traversed, even twice, without ever collecting their profit.

In the following subsection we present a transformation of the original graph that allows us to formulate the CPARP on a complete graph. This transformation has basically two advantages. On the one hand, we will prove that in the transformed graph there exist optimal solutions that traverse no edge of any non serviced component. Later on, this dominance relation will be expressed as a family of inequalities that will reinforce the formulation of the problem. The second advantage is that we will characterize the set of edges that can be traversed twice in any optimal solution. This will result in a considerable reduction on the number of variables that we will define to represent the second traversal of edges.

3.2.1 Graph transformation: CPARP on K_n

Let us consider the complete graph K_n, with the set of vertices $V(D) \cup \{d\}$, so $n = |V(D) \cup \{d\}|$ and $m = n(n-1)/2$. Observe that $V(D) \cup \{d\} = \cup_{k \in \{0,\dots,p\}} V_k$.

The set of demand edges in the transformed problem on K_n is the same of that of the original problem on G. Therefore, also the clusters in K_n will be identical to those in G.

For each edge $e \in E(K_n)$, let P^G_e denote the minimum cost path in G between the end-vertices of e, and therefore, $c(P^G_e)$ its cost. The profit and cost values of the edges of K_n are defined depending on their nature.

- When $e \in D$, let $\Delta_e = c_e - c(P^G_e)$. Then the new cost on K_n is defined as $c_e := c_e - \Delta_e$, and its profit $b_e := b_e - \Delta_e$. This is to say that when $c(P^G_e) = c_e$, e inherits from G both its profit and cost values, b_e and c_e. Otherwise, the cost of e in K_n is defined to be $c(P^G_e)$, thus being reduced in $\Delta_e > 0$. Therefore its profit is defined as $b_e - \Delta_e$. Observe that when an optimal solution to the $CPARP(G, D, d, b, c)$ serves an edge $e \in D$ with $c(P^G_e) < c_e$, then its contribution to the objective function, $b_e - c_e$, is the same than the net profit of edge e in the $CPARP(K_n, D, d, b, c)$. Note also that when $c(P^G_e) < c_e$, it is better to use the minimum cost path that connects the end-vertices of the edge e, than a second traversal of edge e. Thus, such edges will never be used twice in an optimal solution to the $CPARP(G, D, d, b, c)$.
- Otherwise, when $e \notin D$, its cost in K_n is defined to be $c(P^G_e)$ and its

profit, zero.

Proposition 3.1 *The complete graph K_n satisfies the triangular inequality with respect to the costs c.*

$$c_{uv} \leq c_{uw} + c_{wv} \quad \forall u, v, w \in (V(D) \cup \{d\}).$$

Proof *The result follows by the definition of the cost function on K_n.* □

Proposition 3.2 *There is an optimal solution to $CPARP(K_n, D, d, c, b)$ that defines an optimal solution to $CPARP(G, D, d, c, b)$ with the same optimal value.*

Proof *Let $\mathcal{T}^*_{K_n}$ be an optimal solution to $CPARP(K_n, D, d, c, b)$, and let $\mathcal{T}^*_G$ denote the solution to $CPARP(G, D, d, c, b)$, where all edges $e \notin D$ of $\mathcal{T}^*_{K_n}$ have been substituted by their corresponding paths P^G_e. Since $\forall e \notin D$, the contribution to the objective function is $-c_e = -c(P^G_e)$, the value of $\mathcal{T}^*_{K_n}$ and that of $\mathcal{T}^*_G$ coincide. Therefore, $\mathcal{T}^*_G$ is also an optimal solution to $CPARP(G, D, d, c, b)$.* □

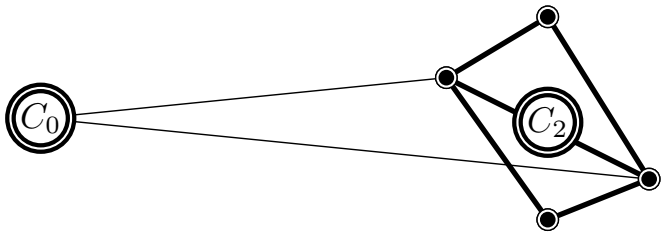

Figure 3.7: *Optimal solution to the instance of Figure 3.5 when the problem is stated on K_n.*

Figure 3.7 shows an optimal solution to the instance of Figure 3.5, when the problem is stated on the complete graph. As can be seen, this optimal solution does not traverse any edge of a component that is not serviced.

Once assumed that throughout we state the CPARP on the complete graph K_n, we modify the notation. From now on, we will denote by V the set of vertices $V := V(D) \cup \{d\}$. Hence, $n = |V(D) \cup \{d\}|$. Consequently, $E := E(K_n)$. Also, we denote by b and c the profit and cost functions on K_n.

3.2.2 Edge Classification for the CPARP

The CPARP will be formulated in terms of a integer linear programming (ILP) problem. In order to do that, we must define the variables involved. In this

subsection we establish several subsets of edges of the CPARP stated on K_n.

First, a partition of the whole set of edges of K_n consisting of three subsets is postulated, depending on the D-parity of their vertices. Then, making reference only to the non-demand edges, another partition consisting of two subsets is established depending on whether or not they connect different clusters. Finally, a singular subset is also defined. Next, we classify edges depending on three different criteria.

According to the D-parity of its vertices: D_e, D_o and D_m

We have three sets of edges that partition the whole set $E(K_n)$: D_e contains all the edges that connect two D-even vertices, D_o contains all the edges that connect two D-odd vertices, and D_m contains all the edges that connect one D-even vertex with one D-odd vertex.

- D-even edges, $D_e = \{uv \in E(K_n) \mid u \text{ D}{-}\textit{even} \text{ and } v \text{ D}{-}\textit{even}\}$
- D-odd edges, $D_o = \{uv \in E(K_n) \mid u \text{ D}{-}\textit{odd} \text{ and } v \text{ D}{-}\textit{odd}\}$
- D-mixed edges, $D_m = E(K_n) \setminus \{D_e \cup D_o\}$

According to the clusters it connects: H_1 and H_2

Throughout we will denote H to the set of non-demand edges $E(K_n) \setminus D$. Given an edge $uv \in H$, we will refer by C_{k_u} and C_{k_v} to the clusters containing vertices u v, being $k_u, k_v \in \{0, \ldots, p\}$.

We have two subsets that partition H:

- Internal edges, $H_1 = \{uv \in H \mid C_{k_u} = C_{k_v}\}$
- Connectivity edges, $H_2 = \{uv \in H \mid C_{k_u} \neq C_{k_v}\}$

Edges in $D_e \cap H_2$ of lowest cost: M_0

For each pair of clusters (C_{k_1}, C_{k_2}), $0 \leq k_1 < k_2 \leq p$, both with some D-even vertex, consider the set of edges $M_{k_1,k_2} = \delta(C_{k_1} : C_{k_2}) = \delta(C_{k_1}) \cap \delta(C_{k_2})$. These are the edges connecting both clusters.

Among them, consider the subset of edges of $M'_{k_1,k_2} \subseteq M_{k_1,k_2}$ that have both vertices D-even, thus $M'_{k_1,k_2} = M_{k_1,k_2} \cap D_e$.

Then, M_0 contains an edge of each non-empty $M'_{k_1k_2}$. Such an edge is of minimum cost among the ones in $M'_{k_1k_2}$ (with ties arbitrarily selected). We do so by selecting the edge e_i with the first edge index of those e_j that have minimum cost in M'_{k_1,k_2}. The expression is

$$e^{k_1,k_2} = \{e_i \in M'_{k_1,k_2} | \forall e_j \in M'_{k_1,k_2}, j > i, c_{e_j} > c_{e_i}\}$$

Once defined e^{k_1,k_2} it is easy to define the set M_0,

$$M_0 = \{e^{k_1,k_2}, 0 \leq k_1 < k_2 \leq p\}. \tag{3.1}$$

Observe that, denoting by α the number of clusters with some D-even vertex, $0 \leq \alpha \leq p$, we have $|M_0| = \alpha(\alpha - 1)/2$.

A summary of all edge sets defined so far is displayed in Figure 3.8.

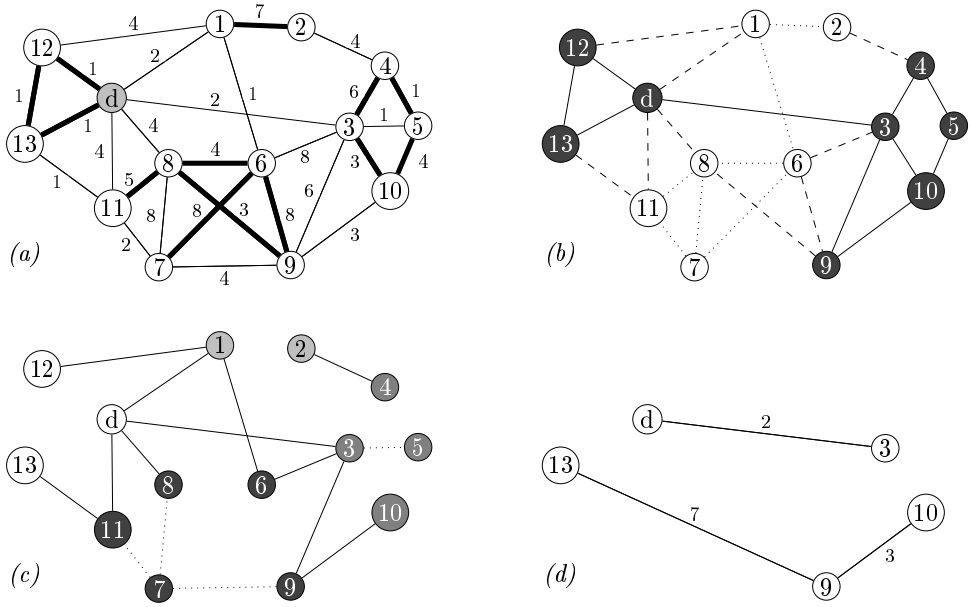

Figure 3.8: *Summary of the definition of sets:* (a) *Instance of CPARP with four clusters. Demand edges in bold (only the costs are shown);* (b) *Vertices D-even in black, vertices D-odd in white, edge set D_e in solid lines, D_m in dashed and D_o in dotted;* (c) *Vertices of V_0 in white, V_1 in light, V_2 in dark and V_3 in black, edge set H_1 in dotted lines, H_2 in solid;* (d) *The three edges that define M_0.*

3.2.3 Properties of the CPARP

Next we study some properties of the CPARP that we will use later on to define a tight formulation of the problem. We distinguish between two types of properties:

- *Feasibility Conditions*: Those that must be satisfied by any feasible solution to the problem, and, thus, split the solution space in either feasible or unfeasible solutions.
- *Dominance Relations*: These propositions need not be satisfied by all feasible solutions, but there is at least one optimal solution that satisfies them. When these relations can be used to discard some edges from making part of an optimal solution we make reference to them as *Preprocessing Statements*.

Feasibility Conditions

There are two types of conditions that are common to feasible solutions of all routing problems and, in particular, to all arc routing problems. On the one hand, from Euler theorem [50] we know that in an optimal solution any vertex must have an even number of incident edges, so it must have even degree. We will refer to these feasibility conditions as *parity restrictions*. On the other hand, in any feasible solution all visited vertices must be connected with the depot someway. Therefore, to prevent solutions from being disconnected, we must also impose the *connectivity restrictions* that also are feasibility constraints.

In addition to these two types of conditions, feasible solutions for the CPARP must satisfy some additional conditions which are specific for the problem. That is, conditions that guarantee that feasible solutions service either all demand edges of the clusters or none of them.

Finally, even though they can not be expressed as inequalities, the *integrality conditions* must also be considered when defining the polyhedron of feasibility of the solution tours.

Dominance Relations

Some dominance relations are presented next. The first two do not need the problem be stated on the complete graph. However from the third, this condition is required. Then, we state the problem on the complete graph with the transformation given in Subsection 3.2.1. The first dominance relation is inherited from the PRPP, see [11].

Dominance 3.3 *There is an optimal solution to the $CPARP(G, D, d, b, c)$ where no edge is used more than twice.*

Proof *We prove this by contradiction. Suppose an edge $e \in E$ were traversed more than two times in an optimal solution $\mathcal{T}^*$. Then, the solution $\mathcal{T}^{**}$ that results by removing from $\mathcal{T}^*$ two copies of edge e is also feasible since it keeps both the parity of the vertices and the connectivity with the depot. Given that $\mathcal{T}^{**}$ has at least one copy of e, we can assume that if $e \in D$ its profit is collected. Finally, since the value of the new solution is strictly better than that of $\mathcal{T}^*$, the solution $\mathcal{T}^*$ could not be optimal. This contradiction arise from supposing that such an edge e could exist.* □

Preprocessing 3.4 *There is an optimal solution $\mathcal{T}^*$ to the $CPARP(G, D, d, b, c)$ in which no edge $e \in \gamma(V_k) \setminus C_k$ is used more that once.*

Proof *Suppose $\mathcal{T}^*$ be an optimal solution to the $CPARP(G, D, d, b, c)$ such that there existed $e = uv \in \gamma(V_k) \setminus C_k$ which was traversed two times in $\mathcal{T}^*$. Observe that removing two copies of e from $\mathcal{T}^*$ would not disconnect C_k from the depot and would not change the parity of u and v. Thus, removing two copies of e from $\mathcal{T}^*$ results in a feasible solution with an objective function value strictly better than that of $\mathcal{T}^*$. Hence, $\mathcal{T}^*$ could not be optimal. This contradiction arise from supposing that such an edge e could exist.* □

Proposition 3.5 *There is an optimal solution to $CPARP(K_n, D, d, b, c)$, $\mathcal{T}^*$, that does not contain two consecutive pairs of parallel edges.*

Proof *Suppose $\mathcal{T}^*$ contains a sequence of parallel edges $\{v_0, v_1\} = e_1 - e_2 - \cdots - e_r = \{v_{r-1}, v_r\}$. By definition of the cost function in K_n, the second "copy" of the sequence can be substituted by edge $\{v_0, v_r\}$ without changing the objective function value.* □

Dominance 3.6 *Let $\mathcal{T}^*$ be an optimal solution to the $CPARP(K_n, D, d, b, c)$ and let $e \in \delta(V_k) \cup (\gamma(V_k) \setminus C_k)$ for some $k \in \{0, \ldots, p\}$. This is, a non-demand edge incident to some cluster. Then, if e is used in the solution $\mathcal{T}^*$, the cluster C_k is serviced by $\mathcal{T}^*$.*

Proof *Suppose that some optimal solution $\mathcal{T}^*$ traverses some edge $e \in \delta(V_k) \cup (\gamma(V_k) \setminus C_k)$, of a cluster k that is not serviced. Then, e makes part of a minimum cost path P_f^G in G for some $f = uv$ between two vertices, u and v, in different (serviced) components (see Figure 3.6). Since P_f^G can be substituted by its associated edge $f = uv \notin D$ connecting different clusters (see Figure 3.7), there exists a feasible solution to $CPARP(K_n, D, d, c, b)$ with the same objective function value than $\mathcal{T}^*$ that does not traverse any edge of a component that is not serviced.* □

Preprocessing 3.4 can be strengthened when the problem is stated on K_n, re-

sulting Preprocessing 3.7.

Preprocessing 3.7 *There is an optimal solution $\mathcal{T}^*$ to $CPARP(K_n, D, d, c, b)$ such that if $uv \in H_1$ is used, then both u and v are D-odd.*

Proof *Suppose that $\mathcal{T}^*$ contains an edge uv such that $uv \in \gamma(V_k) \setminus C_k$ for some k, with u D-even. Therefore, $\mathcal{T}^*$ must also contain, at least, another edge $uw \notin D$, $w \neq v$, to recover the even parity of u. Thus, edge uv is part of a sequence of consecutive edges that defines the path P_e^G, which can be substituted by its associated edge $e = vw$.* □

A very similar argument can be used to prove the following result.

Preprocessing 3.8 *There is an optimal solution to the CPARP such that if $e = uv \in D$ is used twice, then u and v are D-odd.*

Preprocessing 3.9 *There is an optimal solution to $CPARP(K_n, D, d, b, c)$ where the only edges that are used two times are either $e \in D \cap D_o$ or $e \in H_2 \cap D_e$.*

Proof *On the one hand by Preprocessing 3.4, among the ones that connect two vertices in the same component, only edges in D need to be considered as candidates to be traversed twice. And in particular, by Preprocessing 3.8, only edges in D_o are these candidates.*

On the other hand, among the edges that connect different clusters, so $e \in H_2$, suppose that $e \notin D_e$ is used twice in an optimal solution $\mathcal{T}^*$ to the problem on K_n. Given that $e = uv$ has some D-odd vertex, say vertex u, if edge e is used twice in $\mathcal{T}^*$ the parity for vertex u must be restored somehow. Thus, there must exist $f = uw \in \mathcal{T}^*$ and $f \notin D$. Therefore, by Proposition 3.5, we can substitute in $\mathcal{T}^*$ the second traversal of e and the traversal of f by the corresponding edge vw. This new solution would also be optimal and edge e would be traversed only once. □

Finally, the second set of the Preprocessing 3.9 is reinforced by Preprocessing 3.10. Recall that M_0 was defined in Expression 3.1 as the set of edges connecting two different clusters through D-even vertices, that moreover have the minimum costs among all other edges connecting the same pair of clusters through D-even vertices.

Preprocessing 3.10 *There is an optimal solution to $CPARP(K_n, D, d, b, c)$ where the only non-demand edges that are used two times are edges in M_0.*

Proof *Suppose that $\mathcal{T}^*$ is an optimal solution to the CPARP that uses twice some edge $e_1 = u_1v_1 \in H_2 \cap D_e$, thus D-even connecting different clusters. If*

some other edge $e_2 = u_2v_2 \in H_2 \cap D_e$ connecting the same clusters with a lower cost existed, then we could improve the value of $\mathcal{T}^*$ by using twice e_2 instead of e_1. Therefore $\mathcal{T}^*$ could not be optimal. This contradiction arises from supposing that such an edge e_2 could exist. □

3.3 Definition of Variables

In our model variables are associated with clusters and with only those edges that can make part of optimal solution tours. We will distinguish among three types of variables.

3.3.1 Variables associated with the clusters, z_k's

They will indicate whether or not the cluster C_k, $k \in \{0 \dots p\}$, is serviced in the solution. Formally, $z_k = 1 \Leftrightarrow (e \in C_k \Rightarrow e \in \mathcal{T})$, where $\mathcal{T}$ is the solution. Observe that $z_k \in \{0, 1\}$, $k \in \{0 \dots p\}$.

3.3.2 Variables associated with the edges, x_e's and y_e's

We next define the sets of decision variables associated with edges of the complete graph K_n. Let $m = |E(K_n)| = n(n-1)/2$. We define two sets of variables $x, y \in \{0, 1\}^m$ to represent the first and second traversal of edges, respectively. We apply the properties of the previous section to see whether or not for a given edge $e \in E(K_n)$ it is necessary to define associated variables x_e and y_e. Let $E_x \subset K_n$ and $E_y \subset E_x$ denote the sets of edges for which variables x_e and y_e are defined. Their expressions follows:

$$E_x = \{D \cup (H_1 \cap D_o)\} \cup H_2 \tag{3.2}$$

$$E_y = \{D \cap D_o\} \cup M_0. \tag{3.3}$$

The first set composing E_x in (3.2), is related to internal edges to the clusters. As a consequence of Preprocessing 3.7, among them, candidate edges to make part of an optimal solution are, besides demand edges, those edges connecting D-odd vertices, i.e. $(H_1 \cap D_o)$. The second term of (3.2) corresponds to connectivity edges.

With respect to the set E_y, recall from Preprocessing 3.9 that the only candidate demand edges to be crossed twice within the same cluster are edges

connecting two D-odd end-vertices. These are edges in $D \cap D_o$. The last part of Expression (3.3) is a direct consequence of the Preprocessing 3.10.

3.4 ILP Formulation

In this section we present a formulation for the $CPARP(K_n, D, d, b, c)$ that uses the variables that we have defined. We begin by defining the objective function. After that, several families of inequalities are also explained and an ILP formulation for the CPARP is presented.

3.4.1 Objective Function

The objective function must be maximized.

The net profit obtained for servicing all the demand edges of a cluster C_k, for $k \in \{0, \ldots, p\}$ can be expressed as

$$f_k = \sum_{e \in C_k} (b_e - c_e)$$

Then, the expression for the objective function is

$$\max \; Z = \sum_{k=0}^{p} f_k z_k - \sum_{e \in E_x \setminus D} c_e x_e - \sum_{e \in E_y} c_e y_e \tag{3.4}$$

The first term accounts for the net profit of serviced clusters, the second term is the cost of the first traversal of non-demand edges, and the third term corresponds to the second traversal of edges.

3.4.2 Cocircuit Inequalities

Most of problems with binary variables require the parity of vertices. For instance, the 2-matching problem, whose solutions are given by the disjoint union of circuits, as well as different types of routing problems, where solutions are also made of circuits. It is well known that the parity of the vertices can be modelled by means of the so called *cocircuit*, *2-matching* or *blossom* inequalities.

Cocircuit inequalities were formulated by Barahona and Grötschel [17]. Edmonds [47] proved that these inequalities fully characterize the 2-matching polytope, and Grötschel and Holland [62] used them in a cutting plane algorithm

for the perfect 2-matching problem. Given that the 2-matching problem defines one of the most studied relaxations of the Traveling Salesman Problem, since the late eighties Inequalities (3.5) have also been considered for the TSP in the works by Grötschel and Padberg [63] and by Padberg and Rinaldi [94].

In their general form these inequalities are

$$x(\delta(S) \setminus F) \geq x(F) - |F| + 1,\ S \subset V, F \subseteq \delta(S), |F|\ \textit{odd}. \tag{3.5}$$

These inequalities are also facets of the polytope of Eulerian graphs [103] and of the more general polytope of binary matroids [17]. Since the solutions to arc routing problems are given by Eulerian graphs, they have also been considered in cutting plane algorithms in the context of arc routing problems. For instance, Belenguer and Benavent [18] have used them for the Capacitated Arc Routing Problem. Also, Aráoz, Fernández and Meza [9] for the Prize-collecting Rural Postman Problem, and Benavent et al. [24] for the Windy Rural Postman Problem with K vehicles. Ghiani and Laporte [58] have considered the particular case when S is a singleton in the context of the Rural Postman Problem.

Inequalities (3.5) can be adapted to the CPARP with the defined x and y variables resulting in the expression

$$\begin{aligned} x(\delta(S) \setminus F) + y((F \cap E_y) \setminus L) \geq x(F) + y(L) - (|F| + |L|) + 1, \\ S \subset V,\ F \subseteq \delta(S) \cap E_x,\ L \subseteq F \cap E_y, \\ (|F| + |L|) \text{ odd}. \end{aligned} \tag{3.6}$$

The number of inequalities (3.6) is exponential on several variables. In fact, it is in $\Omega(2^n)$, since only taking into account the number of vertices, n, the number of possible subsets S of vertices is $\mathcal{O}(2^n)$.

3.4.3 Connectivity Constraints

Typically, connectivity constraints are formulated by means of inequalities requiring that the cut-set associated with each subset of visited vertices be greater than or equal to two. This is,

$$x(\delta(S)) \geq 2, \qquad S \subset V,\ \ S \neq \emptyset. \tag{3.7}$$

As it is well known, imposing such conditions for each cut-sets of each subset of vertices require a number of inequalities bounded by $\mathcal{O}(2^n)$. In the particular case of the CPARP, the clustering constraint reduces the complexity to $\mathcal{O}(2^p)$ serviced clusters instead of visited vertices. Therefore, the number of connectivity inequalities in the formulation, although it is exponential on p, it is smaller than the number of cocircuit inequalities, since this latter is exponential on a greater number of parameters that moreover have greater values.

For adapting Inequalities (3.7) to the CPARP we note that the only clusters that must be connected with the depot are the ones that are serviced. Thus we obtain,

$$x(\delta(S)) + y(\delta(S) \geq 2z_k, \qquad S = \underset{k\in\pi}{\cup} V_k,\ \pi \in \mathcal{P}(\Omega_p) \tag{3.8}$$

where,

- $\mathcal{P}(\Omega_p)$ is the set of parts of $\Omega_p = \{1, \ldots, p\}$.
- π is any subset of integer numbers from 1 to p. These integer numbers denote cluster indices, thus π represents any subset of clusters not containing the depot cluster.

For a given $\pi \in \mathcal{P}(\Omega_p)$ we have $|\pi|$ inequalities. Observe that the only difference among them is in the right hand side term, that for each inequality is $2z_k,\ k \in \pi$. Given that the empty solution as well as the solution that only services C_0 are feasible, we cannot impose any connectivity constraint on the cut-set $\delta(V_0)$.

3.4.4 ILP Formulation

An integer linear programming formulation, for the CPARP is presented next.

$$(M) \qquad \max \quad \sum_{k=0}^{p} f_k z_k - \sum_{e\in E_x\setminus D} c_e x_e - \sum_{e\in E_y} c_e y_e \tag{3.9}$$

$$x(\delta(S)\setminus F) + y((F\cap E_y)\setminus L) \geq x(F) + y(L) - (|F| + |L|) + 1, \tag{3.10}$$
$$S \subset V,\ F \subseteq \delta(S)\cap E_x,\ L \subseteq F \cap E_y,$$
$$(|F| + |L|) \text{ odd}$$

$$x(\delta(S)) + y(\delta(S)) \geq 2z_k, \qquad S = \cup_{k\in\pi} V_k,\ \pi \in \mathcal{P}(\Omega_p) \tag{3.11}$$

$$x_e = z_k, \qquad e \in C_k, k \in \{0, \ldots, p\} \tag{3.12}$$

$$x_e \leq z_k, \qquad e \in ((\gamma(V_k)\setminus D) \cup \delta(V_k)), k \in \{0, \ldots, p\} \tag{3.13}$$

$$y_e \leq x_e, \qquad e \in E_y \tag{3.14}$$

$$z_k \in \{0, 1\},\ (k \in K) \tag{3.15}$$

$$x_e \in \{0, 1\},\ (e \in E_x) \tag{3.16}$$

$$y_e \geq 0,\ (e \in E_y) \tag{3.17}$$

where $\Omega_p = \{1, \ldots, p\}$.

- Inequalities (3.10) have been seen in Expression (3.6) of Subsection 3.4.2.
- Constraints (3.11) correspond to the connectivity restrictions shown in Expression (3.8) of Subsection 3.4.3.

- Then, Equations (3.12) are the clustering restriction defined for the CPARP.
- Next, Inequalities (3.13) are from Dominance 3.6 in Subsection 3.2.3.
- The result of Dominance 3.3 also in Subsection 3.2.3 follows by Inequalities (3.14).
- Finally, the trivial inequalities for making binary x and z variables.

Next we give a proposition to prove that there is no need to impose y variables to be integer. In order to do that a lema indicating that G_{y^*} contains no tours is proved first.

Throughout, given an optimal solution (x^*, y^*, z^*) to the formulation M, let E_{y^*} be the set of edges of the support graph of the fractional components of y^*. So, $E_{y^*} = \{e \in E \mid 0 < y^*_e < 1\}$ and $G_{y^*} = (V(E_{y^*}), E_{y^*})$.

Lema 3.1 *Let (x^*, y^*, z^*) be an optimal solution to the formulation M. Then, the support graph of the fractional components of y^*, G_{y^*}, is acyclic.*

Proof *Suppose that G_{y^*} had a tour. Let $T_{y^*} \subseteq E_{y^*}$ denote such a tour, and let ϵ denote the minimum value of all y^* values in tour T_{y^*}, $\epsilon = min\{y^*_e \mid e \in T_{y^*}\}$. Therefore, $\epsilon > 0$ by definition of T_{y^*}.*

Let us consider now a new solution (x^{**}, y^{**}, z^{**}) built by making $x^{**} = x^*$, $z^{**} = z^*$, and $y^{**}_e = y^*_e - \epsilon, \forall e \in T_{y^*}$, but $y^{**}_e = y^*_e$ otherwise.

Then, (x^{**}, y^{**}, z^{**}) is feasible for (3.9)-(3.17) and its value is better than that of (x^*, y^*, z^*). Therefore (x^*, y^*, z^*) could not be optimal. □

Proposition 3.11 *Let (x^*, y^*, z^*) be an optimal solution to the formulation M. Then, $y^*_e \in \{0, 1\}, \forall e \in E_y$.*

Proof *Suppose $E_{y^*} \neq \emptyset$.*

By Lema 3.1, there exists $u \in V(E_{y^*})$ which is a leaf in G_{y^*}. This is, $\exists u \in V(E_{y^*})$ such that $\left|\delta_{E_{y^*}}(u)\right| = 1$.

Let $e_u = uv$, be the only edge of E_{y^*} incident with u. Since $\delta_{E_{y^*}}(u) = \{e_u\}$, we have $0 < y^*_{e_u} < 1$.

Now, define the sets $S = \{u\}$ and $F = \{e \in \delta(u) \mid x^*_e = 1\}$. Note that by definition, $F \subseteq \delta(S) \cap E_x$.

Note that, since $x^*_e \in \{0, 1\}, \forall e \in E$, by Inequalities (3.14),

$$0 < y^*_{e_u} \leq x^*_{e_u} \Rightarrow x^*_{e_u} = 1 \Rightarrow e_u \in F.$$

So, we have $\sum_{e \in \delta(u)} x^*_e = \sum_{e \in F} x^*_e = |F| > 0.$

Define the auxiliary set
$L_u = \{e \in \delta(u) \mid y^*_e = 1\}$. *Then we also have that* $\sum_{e \in L_u} y^*_e = |L_u|$.

Now, consider the following two cases depending on the parity of $|F| + |L_u|$*:*

- *If* $|F| + |L_u|$ *is even, then we define the set* $L = L_u \cup \{e_u\}$ *so that* $|L| = |L_u| + 1$ *and thus,* $|F| + |L|$ *is odd.*

  Observe that for these sets F *and* L*,*

  $$x^*(\delta(S) \setminus F) + y^*((F \cap E_y) \setminus L) = 0,$$

  whereas $x^*(F) + y^*(L) - (|F| + |L|) + 1$ *takes the value*

  $$|F| + |L_u| + y^*_{e_u} - (|F| + |L_u| + 1) + 1 = y^*_{e_u},$$

  which is strictly positive.

  Therefore, the cocircuit inequality associated with these sets S, F *and* L *is violated by* (x^*, y^*, z^*)*.*

- *Otherwise* $|F| + |L_u|$ *is odd. Then define the final set* $L = L_u$ *so that* $|F| + |L|$ *is odd.*

  Now, observe that for these sets F *and* L

  $$x^*(\delta(S) \setminus F) + y^*((F \cap E_y) \setminus L) =$$

  $$x(\delta(S) \setminus F) + y((F \cap Ey) \setminus L) = y^*_{e_u} < 1,$$

  whereas

  $$x^*(F) + y^*(L) - (|F| + |L|) + 1 =$$

  $$|F| + |L| - (|F| + |L|) + 1 = 1.$$

  Therefore, also in this case the cocircuit inequality associated with S, F *and* L *is violated by* (x^*, y^*, z^*)*.*

As a consequence, if (x^*, y^*, z^*) *is optimal,* $E_{y^*} = \emptyset$*, and therefore*

$$y^* \in \{0, 1\}, \forall e \in E_y.$$

□

Chapter 4

An Algorithm for the CPARP

In this chapter we present an exact branch and cut algorithm for solving the CPARP. At the root node of the enumeration tree we solve the LP relaxation problem of the formulation M, presented in Chapter 3. Given that this ILP model has an exponential number of constraints we resort to an iterative LP solver scheme where at each iteration the following steps are performed.

1. Solve the current LP relaxation problem of the CPARP instance.
2. If the solution is integer terminate.
3. Solve the separation problem for finding different types of inequalities violated by the solution to the current relaxation.
4. If violated inequalities are found, add them to the current system and jump to step 1, else apply a heuristic and terminate.

Algorithm 4.1 *Main iteration for solving the LP relaxation.*

So, at the end of each iteration we obtain an upper bound to the original problem, given by the solution to the relaxed problem. A flowchart with a general scheme of the Algorithm 4.1 is depicted in Figure 4.1.

Along the next sections, we give a description of the initial relaxation used for the CPARP and we get into details about how separation problems have been solved for the different types of inequalities that have been relaxed. The chapter ends with the description of the exact algorithm. The branching rules

and the criterion used for selecting the candidate subproblem are explained. However, the heuristic used in the exact algorithm will be presented in the next chapter.

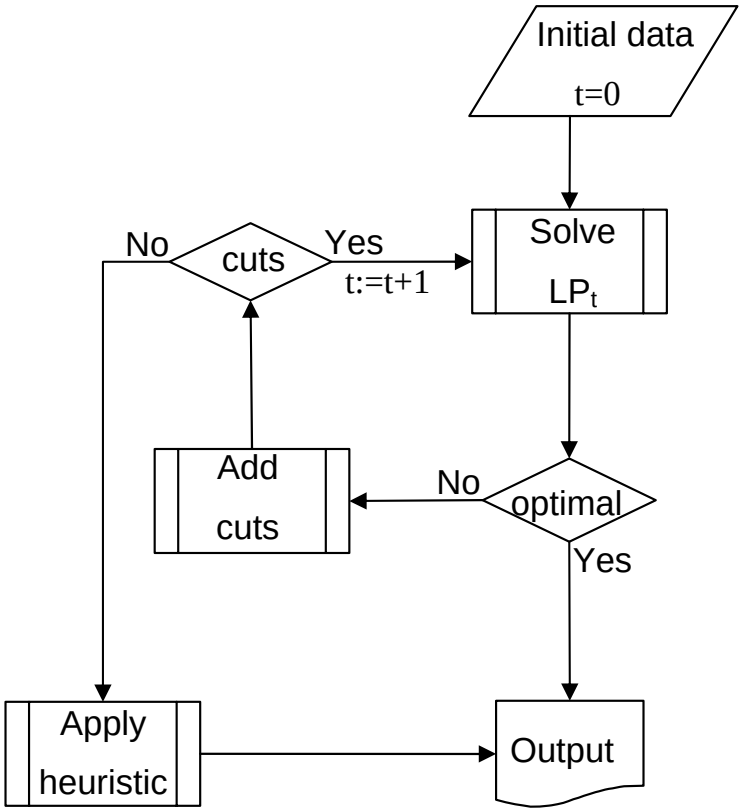

Figure 4.1: *Flowchart of the solution algorithm.*

4.1 Linear Relaxation

For building up an initial linear programming formulation we choose some subset of constraints of the formulation M seen in Chapter 3. In the algorithm, the number of inequalities of each family in the initial LP relaxation has been decided to be in $\mathcal{O}(n)$. Inequalities (3.12)-(3.14) have polynomial sizes and are exactly included in the LP formulation. Instead, only some parity and connectivity constraints are introduces.

On the one hand, with respect to the parity constraints, all cocircuit inequalities (3.10) of the ILP model of the formulation M have been omitted in this initial formulation of the LP relaxation. Instead, we only introduce one equation for each D-odd vertex that we next describe. Thus, it must be expected that all

drawbacks that solutions may carry on about the parity on the D-even vertices will be separated as violated cocircuit inequalities after the first iteration.

On the other hand we select a small subset of connectivity constraints (3.11) in the formulation M. This reduced collection of connectivity restrictions will be seen next, in Subsection 4.1.2.

4.1.1 Initial Parity Inequalities

Cocircuit inequalities (3.10) of the formulation M seen in Subsection 3.4.4 are repeated next.

$$x(\delta(S) \setminus F) + y((F \cap E_y) \setminus L) \geq x(F) + y(L) - (|F| + |L|) + 1, \quad (4.1)$$
$$S \subset V,\ F \subseteq \delta(S) \cap E_x,\ L \subseteq F \cap E_y,$$
$$(|F| + |L|) \text{ odd}.$$

They impose all vertices to have even parity in the solution tours.

The number of such inequalities is in $\Omega(n)$, since just considering those associated with different values of S this lower bound is already reached. So, for the initial LP formulation, Inequalities 4.1 are omitted. Instead, we will only introduce one equation for each D-odd vertex of the graph, so the number of parity restrictions is in $\mathcal{O}(n)$.

We consider the following dominance relations for D-odd vertices.

$$x(\delta(v) \setminus D) + y(\delta(v)) = z_k, \quad v \in V_k, |\delta_D(v)| \text{ odd}, k \in \{0, \ldots, p\} \quad (4.2)$$

These initial parity equations are dominance relations, since the properties imposed by them must be satisfied for all optimal solutions, but other feasible solutions might not fulfill them.

Their meaning is the following. When an optimal solution services a cluster k, thus $z_k = 1$, then for each D-odd vertex $v \in V_k$, the solution must use one additional traversal without profit of some edge incident with v. Getting into more detail, note that this edge might belong to one of the following sets,

- Edges that are traversed once,
  - All non-demand edges with both vertices in the same cluster k. That is, edges in $\gamma(V_k) \setminus C_k$, thus, in H_1. By Preprocessing 3.7, the other vertex of these edges must be D-odd too.

  - All non-demand edges that connect the D-odd vertex v with some vertex of a cluster different from k, thus, in H_2. In this case, the other vertex can be D-even or D-odd.

- Edges that are traversed twice, thus, demand edges with both vertices D-odd, that is $D_o \cap D$, since by Preprocessing 3.8, the only edges in D_o that can be traversed two times in an optimal solution are the edges with demand.

While any feasible integer solution satisfies Constraints (4.2), these equations may not suffice to guarantee the parity of D-odd vertices when integrality conditions on the variables are relaxed, as shown in Figure 4.2.

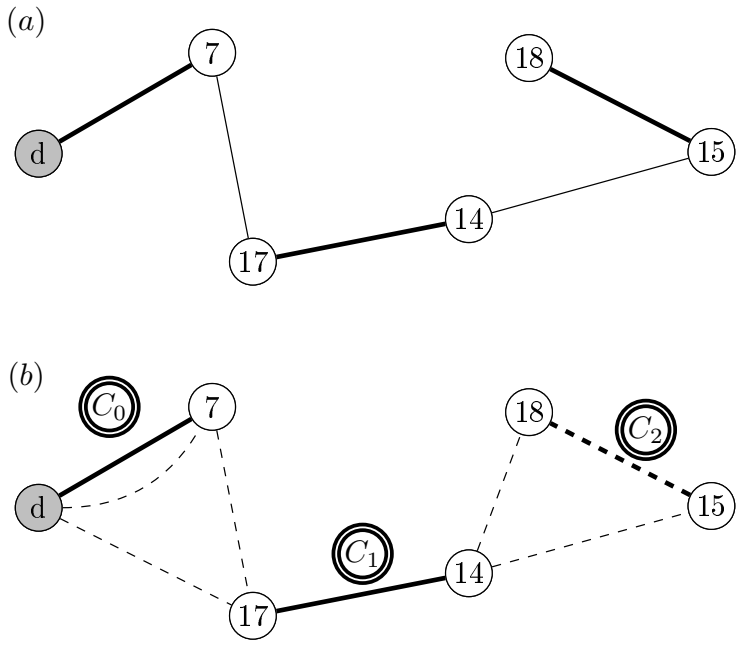

Figure 4.2: (a) *Instance of a CPARP with three clusters. Demand edges appear in bold;* (b) *Unfeasible solution not separated by equations* (4.2)*. Again, demand edges in bold, and dashed lines mean* $x_e = 1/2$*. Other fractional variables are* $y_{d,7} = 1/2$ *and* $z_2 = 1/2$*.*

In Figure 4.2*(a)* an instance of a CPARP with three clusters is depicted, and in Figure 4.2*(b)* an unfeasible solution satisfying Equations (4.2) can be seen. Demand edges are represented with thick lines. In the solution of Figure 4.2*(b)* dashed lines mean edges with solution value $x_e = 1/2$. The cluster C_2 has its corresponding variable $z_2 = 1/2$. Observe that the value $y_{d,7}$ is also equal to $1/2$.

Furthermore, since no parity constraints are introduced in this initial formulation of the LP relaxation for D-even vertices, unfeasible integer solutions

due to violated parity conditions are expected to be found. This is, despite its integrity, they are not Eulerian. In the example of Figure 4.3(a) an instance with all its vertices D-even is depicted.

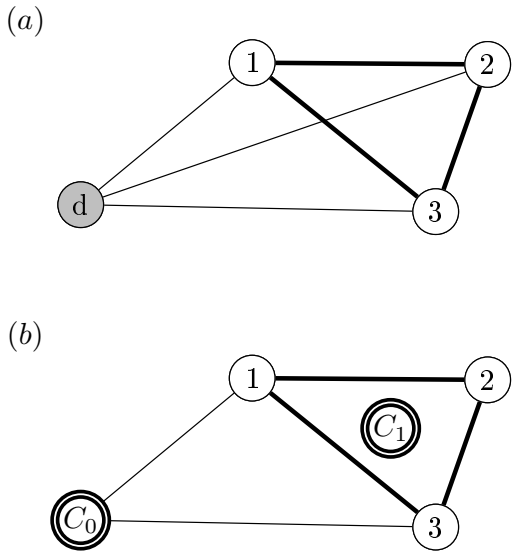

Figure 4.3: (a) *Instance of a CPARP with two clusters. Demand edges appear in bold;* (b) *Unfeasible solution not separated with the initial LP formulation.*

This can be seen in the integral unfeasible solution of Figure 4.3(b).

As said before, it must be expected that unfeasible solutions, like the one shown in Figure 4.3(b), will be separated by the cocircuit separation procedure explained in Subsection 4.2.2.

4.1.2 Initial Connectivity Inequalities

Inequalities (3.11) of the formulation M seen in Subsection 3.4.4 are repeated next.

$$x(\delta(S)) + y(\delta(S)) \geq 2z_k, \qquad S = \cup_{k\in\pi} V_k,\ \pi \in \mathcal{P}(\Omega_p) \tag{4.3}$$

where Ω_p is the set of integer numbers $\{1, \ldots, p\}$, and thus, $\mathcal{P}(\Omega_p)$ the set of all possible subsets of numbers in $\{1, \ldots, p\}$.

They guarantee the connectivity with the depot of any subset of clusters not containing C_0. Then, $\pi \in \mathcal{P}(\Omega_p)$ represents any subset (not containing the value 0) of indices of the clusters.

Inequalities 4.3 exist in a number exponential on p, so we do not introduce all of them in the initial formulation. In particular, we only include those associated with subsets $\pi \in \mathcal{P}(\Omega_p)$ such that $|\pi| = 1$. In this way, the number of connectivity inequalities included in the initial model is in $\mathcal{O}(p)$. The corresponding inequalities can be written as

$$x(\delta(V_k)) + y(\delta(V_k)) \geq 2z_k, \qquad k \in \{1, \ldots, p\}. \tag{4.4}$$

Their meaning is that if a cluster C_k, for $k \in \{1, \ldots, p\}$ (so, different from that of the depot C_0) is serviced, thus $z_k = 1$, then at least two edges of its cut must be traversed too. We do not impose this condition for the depot cluster C_0, since the only solution that just gives service to one cluster that we accept is the one that services the cluster of the depot.

The edges involved in Inequalities must belong to H_2, and therefore, for the y variables, to M_0.

Inequalities (4.4) do not prevent solutions from forming disjoint cycles and, thus, from being disconnected from the depot. It is easy to find solutions that satisfy those inequalities and nevertheless, unfeasible. This is illustrated in the example of Figure 4.4.

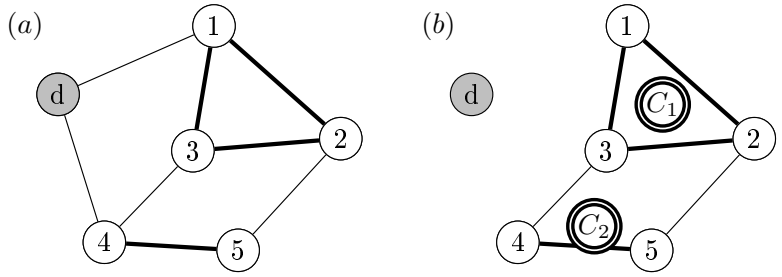

Figure 4.4: (a) *Instance of a CPARP with three clusters. Demand edges in bold;* (b) *Unfeasible solution not separated by Inequalities* (4.4)*. It is servicing clusters* C_1 *and* C_2 *but disconnected from the depot.*

A new instance of the CPARP can be seen In Figure 4.4(a). It has three clusters whose edges are in bold. Figure 4.4(b) illustrates an unfeasible solution (since it is disconnected from the depot) for that instance. As can also be observed, some parity drawbacks exist as well in this solution. The odd parity of its vertices is expected to be separated by the cocircuit separation procedure explained in Subsection 4.2.2.

4.1.3 A Formulation for the Initial LP Relaxation

Before giving the initial formulation for our linear programming (LP) problem, let us recall the definition of some involved parameters.

First, the sets defined in the expressions (3.2) and (3.3), delimiting which variables must be defined.

$$E_x = D \cup \{H_1 \cap D_o\} \cup H_2$$
$$E_y = \{D \cap D_o\} \cup M_0$$

And also, the coefficients f_k introduced in Subsection 3.4.1 that were defined for each cluster k, $k \in \{0, \ldots, p\}$, as

$$f_k = \sum_{e \in C_k} (b_e - c_e).$$

The LP formulation is next depicted.

$$\begin{aligned}
(M0) \qquad \max \quad & \sum_{k=0}^{p} f_k z_k - \sum_{e \in E_x \setminus D} c_e x_e - \sum_{e \in E_y} c_e y_e \\
x(\delta(v) \setminus \delta_D(v)) + y(\delta(v)) &= z_k, \quad v \in V_k,\ |\delta_D(v)| \text{ odd},\ k \in \{0, \ldots, p\} \\
x(\delta(V_k)) + y(\delta(V_k)) &\geq 2z_k,\ k \in \{1, \ldots, p\} \\
x_e &= z_k,\ e \in C_k, k \in \{0, \ldots, p\} \\
x_e \leq z_k, \qquad e &\in ((\gamma(V_k) \setminus D) \cup \delta(V_k)), k \in \{0, \ldots, p\} \\
y_e &\leq x_e,\ e \in E_y \\
z_k &\in [0, 1],\ k \in \{0, \ldots, p\} \\
x_e &\in [0, 1],\ e \in E_x \\
y_e &\geq 0,\ e \in E_y
\end{aligned}$$

4.2 Separation of Inequalities

As already indicated, the LP relaxation problem presented in Subsection 4.1.3 does not contain all the inequalities of the ILP problem of Subsection 3.4.4. Since the goal of the iterative LP based algorithm is to obtain the optimal solution to the LP relaxation problem, the question that we address in this section is how to solve the separation problem for the inequalities of that formulation that have been relaxed.

The strategy that we use for separating relaxed inequalities after obtaining each solution along the iterations, is next explained.

- First, we try to obtain a violated connectivity inequality (4.3).
- If such inequality exists we add it to the current LP and reoptimize the resulting LP.
- If no violated connectivity inequalities exist, then we try to obtain a violated cocircuit inequality (4.1).
- If such inequality is found we add it to the current LP and reoptimize the resulting LP.
- When neither violated connectivity nor cocircuit inequalities are found, the iterative LP based procedure terminates.

Algorithm 4.2 gives a scheme of the separation procedure at each iteration.

```
procedure SEPARATION()
{
    if (not ADD_VIOLATED_CONNECTIVITY()) then
      if (not ADD_VIOLATED_COCIRCUIT()) then
        terminate
      end if
    end if
}
```

Algorithm 4.2 *Separation procedure.*

Observe in Algorithm 4.2 that only when no more violated connectivity inequalities are found in the solution, we try to separate some cocircuit one. Once included the first group of violated cocircuit inequalities, again we will loop until no more violated connectivity exist. Since usually connectivity inequalities involve more variables than cocircuit ones, following this criterion has though to be better.

For the input data of the separation problems we need the value of the solution to the current LP relaxation. So, throughout this section, (x^t, y^t, z^t) denotes the solution to the LP relaxation obtained at some iteration t, $t \in \mathbb{Z}_+$, and $G^t = G_{x^t,y^t,z^t} = (V^t, E^t)$ denotes the subgraph induced by (x^t, y^t, z^t).

As usual, $G^t = (V^t, E^t)$ is defined as follows:

- $E^t = E_{x^t,y^t,z^t} = \{e \in E \mid x_e^t > 0\}$.
- $V^t = V_{x^t,y^t,z^t} = \{u \in V \mid \exists e = uv \in E^t\}$.

4.2.1 Separation of Connectivity Inequalities

Connectivity inequalities (4.3), are repeated here.

$$x(\delta(S)) + y(\delta(S)) \geq 2z_k, \qquad S = \cup_{k\in\pi} V_k,\ \pi \in \mathcal{P}(\Omega_p) \tag{4.5}$$

Next, the separation problem for these inequalities is formally defined.

Definition 4.1 Separation Problem for Connectivity Inequalities
Given a solution (x^t, y^t, z^t) to the LP relaxation problem that induces a graph G^t, to find out whether or not there exists a subset of clusters $\pi_0 \subset \mathcal{P}(\Omega_p)$ and a cluster $k_0 \in \pi_0$ such that

$$x^t(\delta(S_0)) + y^t(\delta(S_0)) < 2z_{k_0}^t, \tag{4.6}$$

where $S_0 = \underset{k\in\pi_0}{\cup} V_k$.

When G^t is disconnected, each connected component defines a violated constraint (4.5). Recall that the connected components of a graph can be obtained with a depth first search algorithm. Many of such algorithms can be found in the literature (T.H. Cormen [42]). The complexity of these algorithms is $\mathcal{O}(n+m)$. In our case, this is $\mathcal{O}(|E^t|)$. However, when the graph G^t is connected, some restriction violated by the current solution (x^t, y^t, z^t) may still exist. From a numerical point of view, connectivity of G^t only guarantees that the left hand side of Inequalities (4.5) be strictly positive. And for being satisfied, it is required not only that the left hand side of the inequality be positive but also be greater than or equal to $2z_k$, for some $z_k > 0$, $k \in \{1, \ldots, p\}$.

The general procedure for separating connectivity inequalities (4.5) was first introduced by Belenguer and Benavent [18] for the Capacitated Arc Routing Problem. The procedure resorts to finding the Gomory and Hu [60] tree of minimum cuts of G^t where the capacities of the edges are given by the values x_e^t. This tree is useful when needing the maxflow solutions for each pair of vertices of a graph. The algorithm of Gomory and Hu uses a *Divide & Conquer* scheme, so it can run in a time bounded by $\mathcal{O}(n)$.

Nevertheless, in our case we are only interested in the connectivity of the serviced components with the depot. And, although we could use the Gomory and Hu algorithm, we can solve the separation problem by only solving the maxflow problems in which C_0 acts as supplier. So, p such problems will be re-

quired in the worst case, giving thus the same complexity than that of Gomory and Hu algorithm. Therefore, the maxflow problems can be solved with the algorithm of Ford and Fulkerson [73]. It consists in an incremental algorithm that works on integer values for the capacities and the flows.

Getting into deeper detail of the separation procedure, we start by defining a new auxiliary graph $M^t(V_m^t, E_m^t)$ with capacities associated with its edges, obtained from the solution graph G^t to the current iteration t. The vertex set V_m^t has just one vertex k for each serviced cluster in G^t. The edge set E_m^t has one edge connecting each pair of vertices of V_m^t. And the capacities of the arcs connecting vertices $k_1, k_2 \in V_m^t$ is given by the sum of the values of the variables connecting both clusters.

1. Construct the graph $M^t(V_m^t, E_m^t)$ with
   - V_m^t containing one vertex k for each cluster C_k such that $z_k^t > 0$.
   - E_m^t contains an edge $e = \{k_1, k_2\}$ for each pair of vertices $k_1, k_2 \in V_m^t$, with capacity
     $$c_e = \sum_{e \in \delta(C_{k_1} : C_{k_2})} (x_e^t + y_e^t)$$
     where $\delta(C_{k_1} : C_{k_2}) \equiv \delta(V_{k_1}) \cap \delta(V_{k_2})$.
2. For each vertex $k \in V_m^t$, $k \neq 0$,
   - Solve the maxflow problem from C_0 to C_k in $M^t(V_m^t, E_m^t)$, obtaining f, the maximum flow at the solution, and π, the set of vertices of V_m^t left in the sink segment of the minimum cut
   - if $f < 2 \ \max\{z_k | k \in \pi\}$, Inequality (4.5) for the set of clusters π is violated by (x^t, y^t, z^t).

Algorithm 4.3 *Algorithm for finding violated connectivity inequalities.*

Algorithm 4.3 is an adaptation of the procedure of Belenguer and Benavent [18] to the case of the CPARP.

The complexity of the procedure detailed in Algorithm 4.3 is closely related to that of the maxflow problems. Denoting $n_t = |V_m^t|$ and $m_t = |E_m^t|$ the complexity of each maxflow problem is in $\mathcal{O}(n_t m_t^2)$, [42]. So, the complexity of the overall procedure is $\mathcal{O}(p^4)$. Empirically, we have found that this bound is

usually almost never reached, since:

- When the maxflow solution from the depot to any other cluster satisfy the inequality, all the clusters used in this solution are satisfying the condition too, and therefore there is no need to solve one special maxflow problem for these other clusters.
- When the maxflow solution from the depot to any other cluster certainly finds out some violated cut, the first cluster in the sink segment of the maxflow solution terminates the loop.

Example

In Figure 4.5 an instance of a CPARP is depicted.

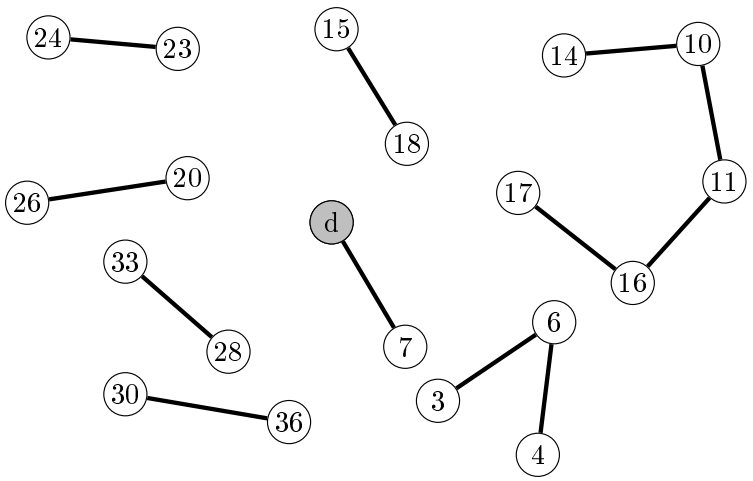

Figure 4.5: *Clusters of an instance of a CPARP.*

For major clarity, only demand edges are shown. Suppose that we want to separate the connectivity inequalities for the unfeasible (fractional) solution depicted in Figure 4.6.

In the solution depicted in Figure 4.6 only three clusters are present, C_0, C_2 and C_3. Demand edges are in bold, dashed lines mean values $x_e = 1/2$. A dashed line also represent the second traversal of edge $\{d, 7\}$ so meaning $y_{d,7} = 1/2$. Note that variable z_3 corresponding to cluster C_3 has a value $z_3 = 1/2$, whereas $z_1 = z_2 = 1$.

Observe that each separate cluster of this solution, but C_0, satisfy the connectivity inequality associated with it. This is, the sum of the solution values in

its cut is greater than or equal (for this case, equal) to the value of the variable z_k associated with it. However, the depot cluster does not satisfy this condition since the constraint is not introduced for it.

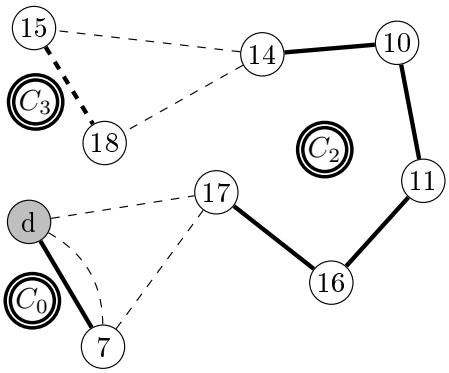

Figure 4.6: *Unfeasible solution $G^t = (V^t, E^t)$ for the instance of Figure 4.5.*

The solution to the maxflow problem posed on the graph M^t associated with the solution of Figure 4.6 is depicted in Figure 4.7. Both capacities are 1.

As can be seen in Figure 4.6 the sum of the variables in the cut-set between C_0 and C_2 is $x_{d,17} + x_{7,17} = 1.0$, which is the value associated to the capacity of the edge $\{C_0, C_2\}$ in the graph M^t. This value coincides thus with the maximum flow between C_0 and C_2. The value of the maximum flow is thus 1, corresponding to edge $\{C_0, C_2\}$, in thick line in Figure 4.7.

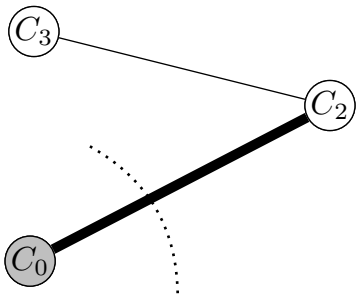

Figure 4.7: *The maxflow problem posed on M^t obtained from G^t of Figure 4.6, with its corresponding solution: edge $\{C_0, C_2\}$ in thicker line.*

Therefore, the maximum flow $f_{0,2} = 1$ in the above cut (composed by the only edge $\{0, 2\}$) is strictly smaller than $2 \max\{z_2, z_3\} = 2$, since $z_2 = 1$. However, z_3 must be considered when selecting this maximum z_k. Observe that C_3 has been left on the sink segment of the minimum cut solution.

Equivalent maximum flow values may be associated with different minimum cut edge sets. This happen whenever two consecutive edges are saturated in the maxflow solution. We always select the edges as far as possible to the depot when facing this kind of solutions. This breaking ties policy makes the whole algorithm faster.

As a consequence Inequality (4.5) associated with $\pi = \{C_2, C_3\}$ is violated by (x^t, y^t, z^t), and therefore, the restrictions

$$\delta(V_2 \cup V_3) \geq 2z_2$$
$$\delta(V_2 \cup V_3) \geq 2z_3$$

added.

4.2.2 Separation of Cocircuit Inequalities

Cocircuit inequalities (3.5) were already presented in Subsection 3.4.2. They are repeated here for convenience.

$$x(\delta(S) \setminus F) \geq x(F) - \mid F \mid +1, \quad S \subset V, F \subseteq \delta(S), |F| \text{ odd} \tag{4.7}$$

Next, the separation of inequalities (4.7) is formally stated.

Definition 4.2 Separation Problem for Cocircuit Inequalities
For a given vector $x^t \in \mathbb{R}^m$, with $0 \leq x_e^t \leq 1$, $e \in E$, to find whether or not exist $S_0 \subset V$ and $F_0 \subseteq \delta(S_0)$, $|F_0|$ odd, such that

$$x^t(\delta(S_0) \setminus F_0) < x^t(F_0) - \mid F_0 \mid +1$$

Inequalities (4.7) can be separated in polynomial time. The basic exact identification procedure is due to Padberg and Rao [93]. Improving techniques for speeding up the convergence of the exact procedure have been proposed by Grötschel and Padberg [63], Grötschel and Holland [62] and Padberg and Rinaldi [94]. These separation algorithms define an extended graph, whose size can raise up to $n + m$ vertices and $2m$ edges, and apply the Padberg and Rao algorithm [93] to it. Hence, applying the Padberg and Rao procedure to such graph results in exact procedures of order $\mathcal{O}((n+m)^4)$, see [63]. Recently, Letchford, Reinelt and Theis [85] have proposed an algorithm of order $\mathcal{O}(n^4)$. Aráoz, Fernández and Meza [9] worked out the basic steps that give an equivalent $\mathcal{O}(n^4)$ separation algorithm for inequalities (4.7). The algorithm is also explained in [6]. The order reduces to $\mathcal{O}M$, when considering only the cases where $|S| = 1$.

Next, we present the exact separation algorithm for inequalities (4.7) and then we particularize if for the CPARP.

Recalling the compact notation stated in Section 2.1 for $x(\delta(s))$, let us point out that Inequalities (4.7) can be rewritten as

$$\sum_{e \in \delta(S) \setminus F} x_e + \sum_{e \in F} (1 - x_e) \geq 1, \quad S \subset V, F \subseteq \delta(S), |F| \text{ odd} \tag{4.8}$$

Therefore, separating cocircuit inequalities mean minimizing the left hand side of Inequalities (4.8).

In what follows we will work with a solution vector $x^t \in \mathbb{R}^m$ with $0 \leq x_e^t \leq 1$, $e \in E$, that induces a graph $G_{x^t} = (V_{x^t}, E_{x^t})$. Each edge has a capacity $h_e = min\{x_e^t, 1 - x_e^t\}$. And also, a partition of E_{x^t} is established, $E_{x^t} = E_- \cup E_+$, where

- $E_- = \{e \in E_{x^t} \mid h_e = x_e^t\} = \{e \in E_{x^t} \mid x_e^t < 0.5\}$
- $E_+ = \{e \in E_{x^t} \mid h_e = 1 - x_e^t\} = \{e \in E_{x^t} \mid x_e^t > 0.5\}$

Edges with value $x_e^t = 0.5$ are assigned arbitrarily to E_- or to E_+. Then, a set of vertices $S \subset V_{x^t}$ is said to be odd^+ if and only if $\left|\delta_{E_+}(S)\right|$ is odd, and $even^+$ otherwise.

The following proposition is proved in [9].

Proposition 4.1 *Let $S \subset V$. The following two properties hold:*

(a) When S is odd^+ then a subset F, $|F|$ odd, that minimizes the value of the left hand side of constraint (4.8) *is given by $F = \delta_{E_+}(S)$. The minimum value of the left hand side of Inequality* (4.8) *is then given by $h(\delta(S))$.*

(b) When S is $even^+$, a subset $F \subseteq \delta(S)$, $|F|$ odd, with minimum value of the left hand side of constraint (4.8) *can be obtained as follows. If $x_{e^1} - (1 - x_{e^1}) < (1 - x_{e^2}) - x_{e^2}$ then $F = \delta_{E_+}(S) \setminus \{e^1\}$, with $x_{e^1} = \min\{x_e \mid e \in \delta_{E_+}(S)\}$. Otherwise, $F = \delta_{E_+}(S) \cup \{e^2\}$, with e^2 such that $x_{e^2} = \max\{x_e \mid e \in \delta(S) \setminus \delta_{E_+}(S)\}$. The minimum value of the left hand side of Inequality* (4.8) *is given by $h(\delta(S)) + \min\{x_{e^1} - (1 - x_{e^1}), (1 - x_{e^2}) - x_{e^2}\}$.*

Throughout for a given cut-set $\delta(S)$, e^1 and e^2 denote arbitrarily chosen edges such that $x_{e^1} = \min\{x_e \mid e \in \delta_{E_+}(S)\}$ and $x_{e^2} = \max\{x_e \mid e \in \delta_{E_-}(S)\}$.

By Proposition 4.1, given a vector x, the smallest value of the left hand side of Inequalities (4.8) may correspond either to an odd^+ set S with $F = \delta_{E_+}(S)$,

or to an $even^+$ set S, with a modified set F defined as indicated in Proposition 4.1(b). Hence, in order to solve the separation problem for a given vector x we must identify both the odd^+ set S with minimum capacity $h(\delta(S))$, and the $even^+$ set S that minimizes $h(\delta(S)) + \min\{x_{e^1} - (1 - x_{e^1}), (1 - x_{e^2}) - x_{e^2}\}$. The result of Padberg and Rao [93] indicates how to identify the odd^+ set S with minimum capacity $h(\delta(S))$. For obtaining these sets, a tree of minimum cuts, between all pairs of vertices of the support graph is used.

1. Obtain T, the tree of minimum cuts of all pairs of odd$^+$ vertices of G_{x^t}.
2. Let $\delta(S^i)$, $i = 1, \ldots, r$ be the cut-sets associated with the edges of T.
3. *for* $i = 1, \cdots, r$ *do*

   3.1 *if* S^i *is* odd^+ *and* $h(\delta(S^i)) < 1$ Terminate.
   (Inequality (4.8) with $S = S^i$ and $F = \delta(S^i)$ is violated by x^t).

   3.2 *if* S^i *is* $even^+$ *and* $h(\delta(S^i)) + \min\{x^t_{e^1} - (1 - x^t_{e^1}), (1 - x^t_{e^2}) - x^t_{e^2}\} < 1$

   then

   if $(x^t_{e^1} - (1 - x^t_{e^1}) < (1 - x^t_{e^2}) - x^t_{e^2}$ *then*
   $F = \delta(S^i) \setminus \{e^1\}$
   otherwise
   $F = \delta(S^i) \cup \{e^2\}$
   endif

   endif

   Terminate.
   (Inequality (4.8) with $S = S^i$ and F is violated by x^t).

   endif

endfor

Algorithm 4.4 *Exact separation algorithm for cocircuit inequalities.*

The exact algorithm proposed in [9] starts building T, the tree of minimum cuts of all pairs of vertices of G_{x^t} relative to the capacities vector h. Then the edges in T are explored in turn until a minimum cut is found for which Inequality (4.8) is violated, or until all the edges in T have been considered. For each considered minimum cut $\delta(S^i)$, the vertex set S^i is either odd^+ or $even^+$. In the first case, if $\left|S^i\right|$ is odd, it only has to be checked whether or not its capacity is smaller than one. If it is smaller, then $S = S^i$ and $F = \delta_{E_+}(S^i)$ solve the separation problem. Otherwise, when $\left|S^i\right|$ is $even^+$, the associated set F will be $F = \delta_{E_+}(S^i) \setminus \{e^1\}$ when $x^t_{e^1} - (1 - x^t_{e^1}) < (1 - x^t_{e^2}) - x^t_{e^2}$, or $F = \delta_{E_+}(S^i) \cup \{e^2\}$ otherwise. Once F is defined, we have to check whether or not Inequality (4.8) is violated b y F together with S^i. The procedure is given in Algorithm 4.4.

Most of the computational burden of Algorithm 4.4 is due to the computations for obtaining the tree of minimum cuts. That is, due to the selection of vertices to establish a candidate vertex set S. However, for the case of the CPARP, a heuristic cocircuit separation has been developed so that no need to compute the tree exist. Next, the specific algorithm for separating Inequalities (4.8) when $|S| = 1$ is described. Later we get into details in the heuristic procedure used in the algorithm for the CPARP, for sets S with more than just one vertex.

Case $|S| = 1$

When S is a singleton, the separation of Inequalities (4.7) can be done in $\mathcal{O}M$ as explained next. Observe that if $S = \{v\}$, these inequalities can be written as

$$x(\delta(v) \setminus F) \geq x(F) - \mid F \mid +1, \quad F \subseteq \delta(v), |F| \text{ odd} \tag{4.9}$$

that also can be rewritten as:

$$\sum_{e \in \delta(v) \setminus F} x_e + \sum_{e \in F} (1 - x_e) \geq 1, \quad v \in V, F \subseteq \delta(S), |F| \text{ odd} \tag{4.10}$$

Then, solving the separation problem for $|S| = \{v\}$ is to find those edges incident with it that minimize the left hand side of Inequality (4.10). For this case, the separation procedure is detailed in Algorithm 4.5 (see [9] for details).

for $v \in V$ *do*

1. $F = \{e \in \delta(v) \mid x^t_e \geq 0.5\}$.
2. *if* $|F|$ *is even then*
   $x^t_{e^1} = \min\{x^t_e \mid e \in F\}$
   $x^t_{e^2} = \max\{x^t_e \mid e \in \delta(v) \setminus F\}$
   if $x^t_{e^1} - 0.5 \leq 0.5 - x^t_{e^2}$ *then*
   delete e^1 from F
   otherwise
   add e^2 to F.
   endif
   endif
3. If the inequality (4.7) corresponding to this set F is violated by x^*, then we have a cut.

endfor

Algorithm 4.5 *Separation algorithm for* $S = \{v\}$.

A proof of the next proposition can be found in [9].

Proposition 4.2 *The separation Algorithm 4.5 is exact and its complexity is in $\mathcal{O}M$.*

Heuristic Cocircuit Separation for the CPARP

Applying the procedure of the exact Algorithm 4.4 at every iteration of a LP solver based algorithm can be quite time consuming. For this reason in our case we use a heuristic separation method for Inequalities (4.7) that we next describe.

Observe that each connected component of the graph induced by the edges with fractional values, $0 < x_e^t < 1$ or $0 < y_e^t < 1$, already defines a set S^i with $\delta(S^i)$ in the tree T used in Algorithm 4.4. The connected components of a graph can be obtained with a much smaller computational burden than the exact algorithm.

It is clear that a very simple and fast heuristic consists in computing the connected components of G_ϵ^t, induced by edges such that $x_e^t > \epsilon$ or $y_e^t > \epsilon$, for a given parameter $0 \leq \epsilon < 1$. Thus, in our heuristic we create an auxiliary graph $B(V_b, E_b)$, based on the current solution values (x^t, y^t, z^t), that will have all vertices of the solution, so $V_b = V^t$, but only those edges $e \in E^t$ with fractional values for x_e or y_e such that $\epsilon < x_e < 1$ or $\epsilon < y_e < 1$. Then, the connected components of B will be used as candidate subsets of vertices S.

We also test all isolated vertices as candidate subsets of vertices S in order to make use of Algorithm 4.5.

In this way we avoid computing the tree of Gomory and Hu needed in the exact separation algorithm.

When adapting Inequalities (4.8) to the CPARP we obtain the expression

$$\sum_{e \in \delta(S) \setminus F} x_e + \sum_{e \in F \setminus L} y_e + \sum_{e \in F} (1 - x_e) + \sum_{e \in L} (1 - y_e) \geq 1 \tag{4.11}$$

$$S \subset V,\ F \subseteq \delta(S),\ L \subset F,\ |F| + |L|\ \ odd$$

Next, a detailed algorithm to establish F and L for a given candidate S is presented in Algorithm 4.6.

Some explanation of the parameters may be convenient. First, note that the procedure is giving back a logical value whose meaning is whether or not a violated cocircuit inequality has been found. Also, observe that three sets are given to the function, S as input data, and F together with L as output.

```
bool SEPARATE_COCIRTUIT(set S, set& F,set& L)
{
    double min_upper = ∞        // minimum variable value > 1/2
    int e+ = -1                 // index of the previous minimum
    double max_lower = 0        // maximum variable value < 1/2
    int e− = -1                 // index of the previous maximum
    F = L = ∅
    foreach e ∈ δ(S) do {
       if (x_e > 0.5) {
          F = F ∪ {e}
          if (min_upper > x_e) {
             e+ = e
             min_upper = x_e
       }} else {
          if (max_lower < x_e){
             e− = e
             max_lower = x_e
       }}
       if (y_e > 0.5) {
          L = L ∪ {e}
          if (min_upper > y_e) {
             e+ = e
             min_upper = y_e
       }} else {
          if (max_lower < y_e){
             e− = e
             max_lower = y_e
    }}}
    if (|F| + |L|) even {
       if (min_upper - 0.5 < 0.5 - max_lower) {
          F ∪ L = F ∪ L \ {e+}
       } else {
          F ∪ L = F ∪ L ∪ {e−}
    }}
    return (EVALUATE_COCIRCUIT(S,F,L) < 1)
}
```

Algorithm 4.6 *Procedure to find edge sets F and L for a given vertex set S.*

Given that for knowing the edges of the set $\delta(S)$ it is required a computational time in $\mathcal{O}(n^2)$, it may be worthy to modify Algorithm 4.6, and with some

lines of code, to store these identifiers somewhere. Doing so will save this time when adding the inequality to the formulation.

Example

Next we present an example of the heuristic procedure used for the separation of cocircuit inequalities. An instance is shown in Figure 4.8(a). Demand edges are in bold. The graph $G^t = (V^t, E^t)$ induced by the fractional solution (x^t, y^t, z^t) can be seen in Figure 4.8(b), with solid lines for edges with $x_e = 1$ and dashed lines for edges with $x_e = 1/2$.

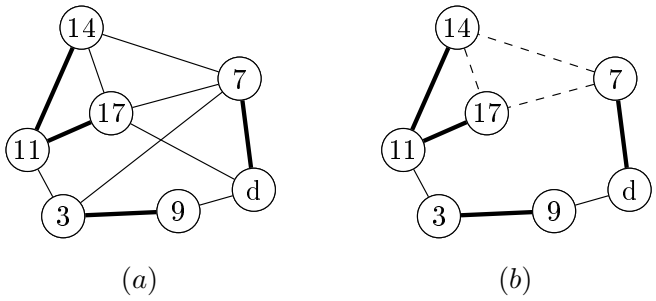

Figure 4.8: *(a) Instance of a CPARP with three clusters; (b) Unfeasible (fractional) solution G^t.*

When $\epsilon = 0$, the connected components of G^t_ϵ are $S^1 = \{11\}$, $S^2 = \{3\}$, $S^3 = \{9\}$, $S^4 = \{d\}$ and $S^5 = \{7, 14, 17\}$. Thus, we call the procedure of Algorithm 4.6 once for each isolated vertex and also for $S = \{7, 14, 17\}$.

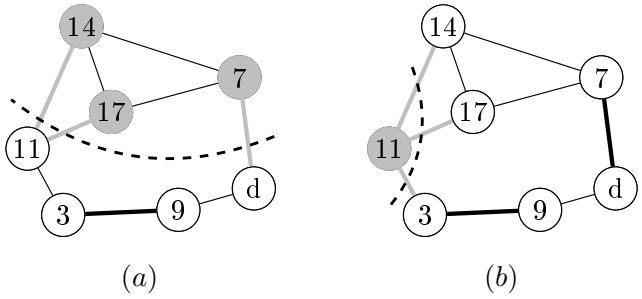

Figure 4.9: *Cuts in G^t of Figure 4.8(b) that do not satisfy cocircuit inequalities* (4.11)*, displayed in light grey thick lines.*

In Figures 4.9(a) and (b), the two violated cocircuit inequalities found by the heuristic can be seen, corresponding to $S = \{7, 14, 17\}$ and $S = \{11\}$.

Empirically we have found that violated cocircuit inequalities are found by pairs. This seems to be related to the Euler theorem about the even number of odd vertices in graphs.

Even though using the exact procedure seen in Algorithm 4.4 would have ensured that no violated Inequality (4.5) existed at the termination of the LP relaxation problem, using a simpler heuristic method as the explained above has been enough. However, it is quite reasonable to continue the research in this line to evaluate the required cpu time if the Gomory and Hu algorithm was used.

4.3 Branch & Cut Algorithm

Algorithm 4.1 terminates when no more violated inequalities of types (4.5) or (4.7) are identified. Then, when the optimality of the last LP solution is not proved we apply a heuristic to obtain a feasible CPARP solution. The heuristic will be presented in Chapter 5.

Let (x^r, y^r, z^r) denote the optimal LP solution and Z^r its associated value. Let also denote (x^h, y^h, z^h) the integer solution obtained with the heuristic method, and Z^h the corresponding value. The optimality of Z^r can be established when the LP solution (x^r, y^r, z^r) is feasible, so integer, for the CPARP. When it is not, the optimality of Z^h can be established if $Z^h \geq \lfloor Z^r \rfloor$. When neither Z^r nor Z^h are proved to be optimal we resort to an exact enumerative algorithm that we describe next.

Recall that by Proposition 3.11 the only variables that can be candidates for branching are the z and x variables. We define a search tree that is separated in two stages, one for each of the above types of variables. Since the z variables have a higher influence on the structure of the problem, we first branch on the z variables until all of them are integer. Then, when needed, we branch on the x variables. A scheme of the search tree is depicted in Figure 4.10.

The criterion to select the variable for branching at any node of the z-levels of stage 1 is described next. Let us denote $b_k = \min\{z_k, 1 - z_k\}\ \forall k \in \{0, \ldots, p\}$. Then, the selected variable for the branch is $z_{\widehat{k}}$ corresponding to the index $\widehat{k}$ such that $b_{\widehat{k}} = \max\, b_k,\ 0 \leq k \leq p$.

Hence, we branch on the z variable that takes the value closest to 1/2. Once the branching variable is selected we proceed depending on different cases.

- If $\min\{z_{\widehat{k}}, 1 - z_{\widehat{k}}\} = z_{\widehat{k}}$, that is when $z_{\widehat{k}} < 0.5$, we explore first the branch $z_{\widehat{k}} = 0$.

- Otherwise, we explore first the branch $z_{\widehat{k}} = 1$.

At any node of the x-levels, in the stage 2 of the tree, we use the same criterion as above, now applied to the x variables.

At both stages of the tree we have tested several more sophisticated criteria both for selecting the branching variable and the first branch from it, although this did not improve our results.

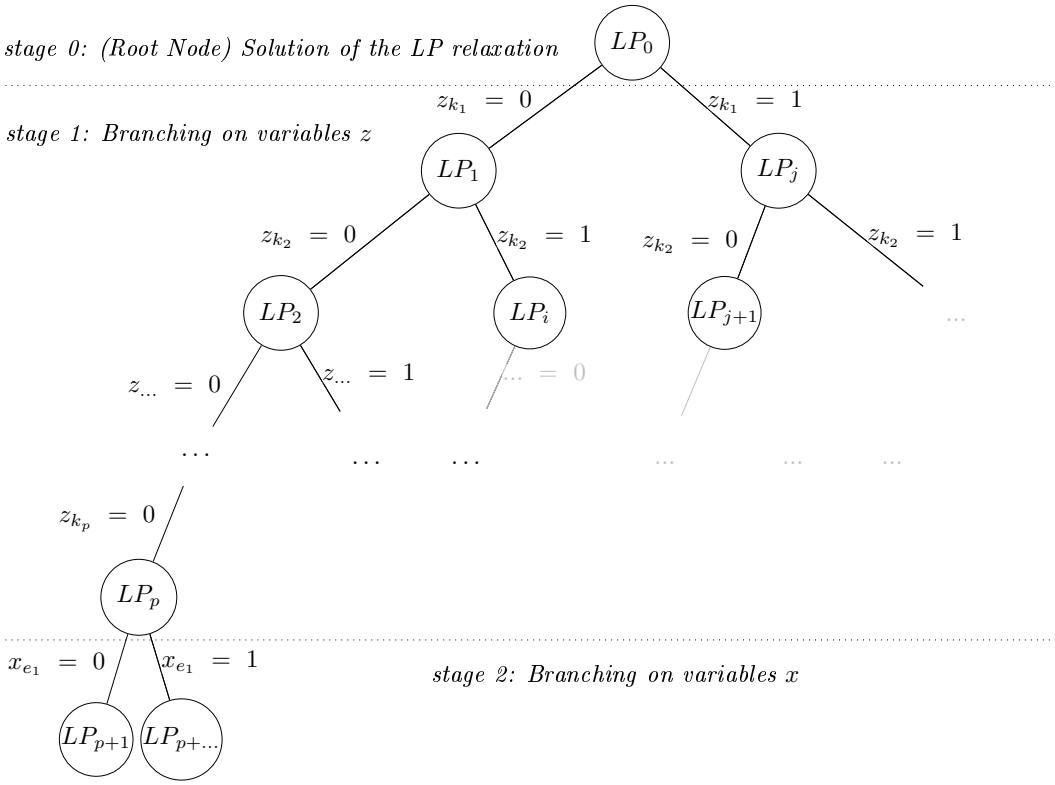

Figure 4.10: *Scheme of the enumeration tree.*

Chapter 5

Heuristic Approaches

In this chapter we present different heuristic approaches to the CPARP. These methods generate feasible solutions and thus lower bounds for the problem. Their quality may be evaluated by means of the upper bounds given by the solution to the LP relaxation presented in Chapter 4. Heuristic methods are also a very useful tool in exact algorithms, since the associated bounds can be used for pruning unpromising branches of the search tree.

The first part of this chapter concentrates of constructive heuristics. We present two different types of such methods. The first type is based on the idea of merging clusters in the most promising ways, whereas in the second type, the constructive heuristic is based on the idea of building feasible solutions starting from a spanning tree through a set of clusters. The second part of the chapter describes improvement methods based on local search that have been considered.

For simplicity of the heuristic methods that we apply, we work again on a complete graph. Therefore, any edge can be used at any stage of the heuristics. However, one might expect that edges that have been eliminated by some dominance relation in Chapter 3 will not appear in the final solutions.

A solution will be represented by a multiset containing all the edges that are serviced or traversed in the solution. When a demand edge is both serviced (with profit) and traversed (without profit) or when a non-demand edge is traversed two times, such an edge will appear twice in the multiset. Throughout, the second traversal (so, without profit) of a demand edge or any traversal of a non-demand edge is referred to as a *non-profit traversal* of an edge.

From an heuristic point of view a difficulty that we have to face when handling the demand clusters is that these components are not necessarily Eulerian. We denote C_k^e, $k \in \{0, \ldots, p\}$ to the Eulerian graphs of minimum cost containing components C_k. It is well-known that these *Eulerian components* can be

found by solving a perfect matching problem on the graph induced by the D-odd vertices of C_k.

The value $c(C_k^e)$ represents the overall cost of the links of $\gamma(C_k)$ in any solution to the CPARP that services cluster k, if the cluster is connected with the rest of the solution by means of any D-even vertex.

In Figure 5.1(a) a star shaped cluster is displayed. Close to edges are their profits in light over their costs in black. Its corresponding C_k^e is shown in 5.1(b). Once the component is made Eulerian we can define the net profit $p_k^e = b(C_k) - c(C_e^k)$. As can be seen in Figure 5.1(b), $c(C_k^e) = 18$, while the net profit $p_k^e = 1$.

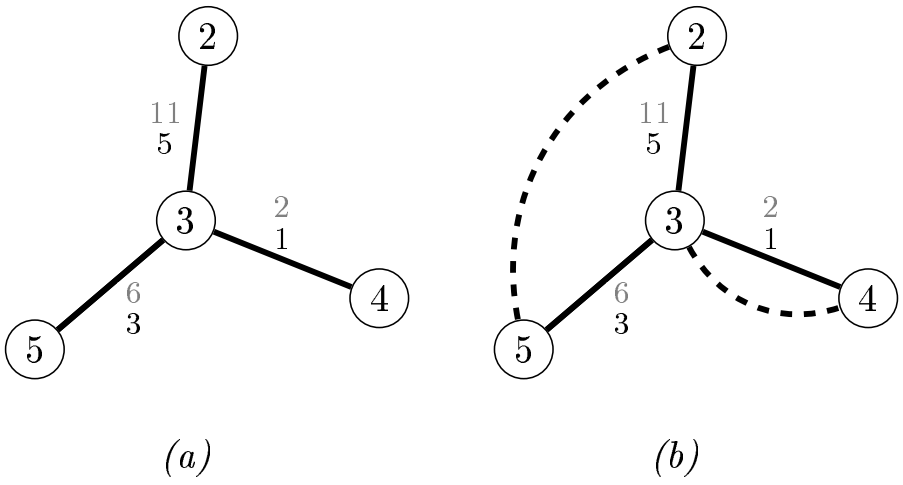

Figure 5.1: (a) *A cluster. Profits in light, costs in black;* (b) *The corresponding set* C_k^e *with cost* $c(C_k^e) = 18$ *and profit* $p_k^e = 1$.

5.1 Constructive heuristics

Several ideas for finding feasible solutions in small computation times have been tested. Next we present two types of constructive heuristics based on simple ideas. Both use different criteria for building solutions that service clusters that seem promising.

5.1.1 Merging clusters heuristic

In this method, not only C_k^e is used. For a given cluster $k \in \{0, \ldots, p\}$ we will also define a $C_k^{u,v}$ for each pair of D-odd vertices $u, v \in V_k$. We denote $C_k^{u,v}$ the

(non-Eulerian) graph of minimum cost containing C_k, in which the only odd vertices are u and v. In this way, the value $c(C_k^{u,v})$ represents the overall cost of the links of $\gamma(C_k)$ in any solution that services cluster k, if the cluster were connected with the rest of the solution by means of u and v.

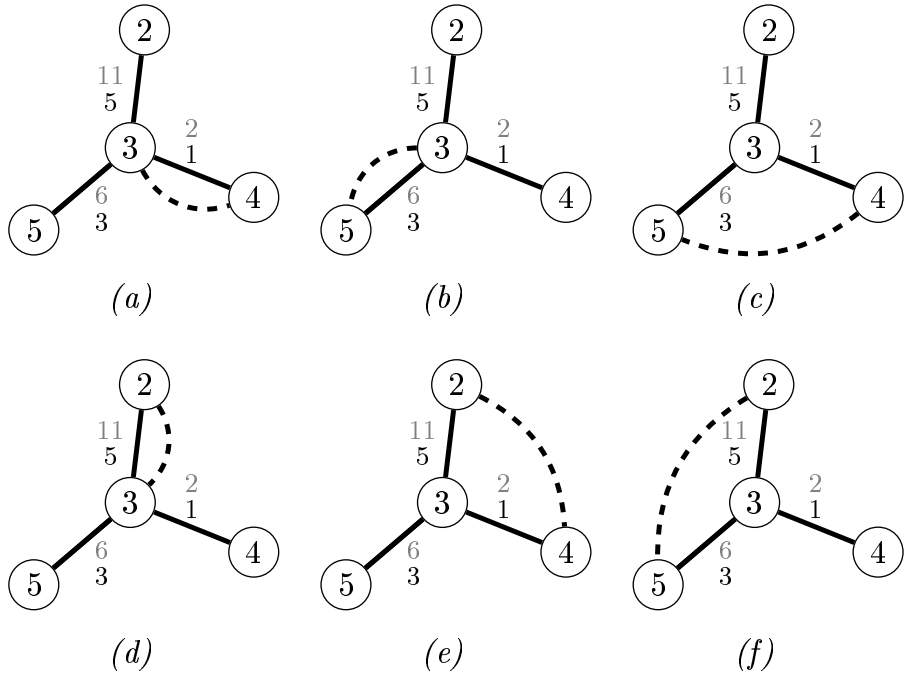

Figure 5.2: *Auxiliary graphs defined from the cluster of Figure 5.1 ordered by profits: (a)* $p(C_k^{2,5}) = 9$*; (b)* $p(C_k^{2,4}) = 7$*; (c)* $p(C_k^{2,3}) = 6$*; (d)* $p(C_k^{4,5}) = 5$*; (e)* $p(C_k^{3,5}) = 4$*; (f)* $p(C_k^{3,4}) = 2$*;*

Figure 5.2 depicts these auxiliary graphs for the example of Figure 5.1. The number of graphs $C_k^{u,v}$ for each cluster is $d(d-1)/2$ being d the number of D-odd vertices of cluster C_k. Also, the associated profit obtained when servicing cluster k using the edges of $C_k^{u,v}$ is $p(C_k^{u,v}) = b(C_k) - c(C_k^{u,v})$.

The heuristic method of merging clusters makes use of a priority queue for each cluster using the above profits as priorities in such a way that C_k^e will be considered only after all the $C_k^{u,v}$ have been discarded.

The main idea of this heuristic is to start from some feasible solution, as for instance the Eulerian cluster C_0^e, and iteratively merge clusters in the solution in a greedy fashion, one at a time, until there are no more improvements. At each iteration we look for the best cluster to be merged with the current solution. Let G^t denote the current solution at iteration t, and let $\pi(G^t)$ denote the set of clusters serviced by G^t. Observe that there are four possibilities for merging a cluster C_k, $k \notin \pi(G^t)$ into G^t:

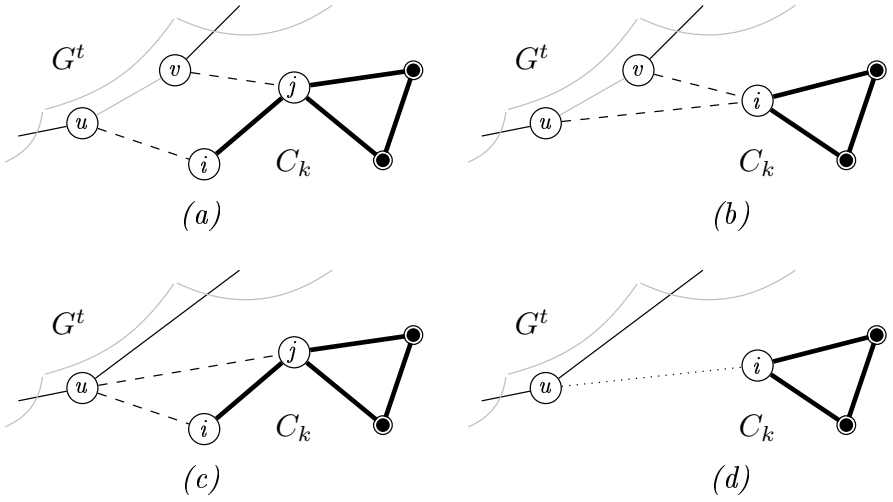

Figure 5.3: *The four possibilities of merging a new cluster C_k in the current solution G^t. (a) The traversal of edge uv in light is substituted by the cluster C_k through its vertices i and j adding the dashed edges to the current solution G^t; (b) The traversal of edge uv in light is substituted by the cluster C_k accessed through the single vertex i twice, introducing the dashed edges; (c) The cluster C_k is inserted (connecting its vertices i and j with vertex u), thus inserting the two edges in dashed lines; (d) Cluster C_k is inserted in the current solution at the vertex u, introducing twice the edge in dotted line.*

1. *merge-edge-odd*: We substitute the traversal without profit of an edge $e = uv \in G^t$ by the cluster $C_k^{i,j}$, not in G^t. The new cluster $C_k^{i,j}$ must be connected with the current solution G^t by means of two edges that connect two different D-odd vertices $i, j \in V_k$ with u and v. That is, for each traversal without profit $e = uv \in G^t$, and for each $k \notin \pi(G^t)$, it is necessary to identify the pair of D-odd vertices $i, j \in V_k$ that give the maximum profit. The solution is updated to $G^t := G^t \cup C_k^{i,j} \cup \{ui, jv\} \setminus \{uv\}$ (see Figure 5.3*(a)*).

2. *merge-edge-even*: We substitute a traversal without profit of an edge $e = uv \in G^t$ by a component C_k^e which is connected with the partial solution by means of two edges that connect a D-even vertex $i \in V_k$ with u and v. That is, for each traversal without profit $e = uv \in G^t$, and for each $k \notin \pi(G^t)$, it is necessary to identify the D-even vertex $i \in V_k$ that gives the maximum profit from the substitution of the traversal of the edge $e = uv$ by component C_k. Once i is identified, the partial solution is updated to $G^t := G^t \cup C_k^e \cup \{ui, iv\} \setminus \{uv\}$ (see Figure 5.3*(b)*).

3. *merge-vertex-odd*: This case is similar to *merge-edge-odd* with the only difference that the two D-odd vertices $i, j \in V_k$ are connected to the same vertex u of G^t, so no edge is removed. Now, $C_k^{u,v}$ is inserted in the solution. That is $G^t := G^t \cup C_k^{i,j} \cup \{ui, ju\}$ (see Figure 5.3*(c)*).

4. *merge-vertex-even*: This case is similar to *merge-edge-even* with the only

difference that the D-even vertex $i \in V_k$ is connected to the same vertex u in G^t with two copies of the edge ui. Again, no edge of G^t is removed, and $C_k^{u,v}$ is inserted. That is $G^t := G^t \cup C_k^e \cup \{ui, ui\}$ (see Figure 5.3*(d)*).

In Algorithm 5.1 there are four calls to procedures corresponding to the four possibilities of merging described above. Each one of them returns a logical value indicating whether or not the solution G^t has been improved due to a successful merging of that type. Parameters in these calls are merely orientative. In the core of each of these four procedures the priority queue for cluster C_k is used extensively.

```
int MERGE_ALL(G^t)
{
   repeat
   {
      for each traversal without profit of edge edge e = uv ∈ E^t)
      {
         for each cluster k ∉ π(G^t)
         {
            HasImproved = MERGE_E_ODD(G^t, e, C_k)
            if (NOT HasImproved)
              HasImproved = MERGE_E_EVEN(G^t, e, C_k)
         }
      }
      if (NOT HasImproved) {
        for each vertex u ∈ V^t
        {
           for each cluster k ∉ π(G^t)
           {
              HasImproved = MERGE_V_ODD(G^t, u, C_k)
              if (NOT HasImproved)
                HasImproved = MERGE_V_EVEN(G^t, u, C_k)
           }
        }
      }
   } while (HasImproved)
   return z(G^t)
}
```

Algorithm 5.1 *Heuristic for merging clusters.*

The kernel of this heuristic is detailed in the Algorithm 5.1. The four functions called in it use the priority queues of cluster C_k in order to find the minimum cost links that can be used to introduce that cluster into the solution. As can be seen, the function receives the current solution as its input parameter.

Several tests have been done with different initial solutions. Not only C_0^e but also all other C_k^e, $k \in \{1, \ldots, p\}$. Observe that in these latter cases, some action must be taken when the depot does not appear on the final solution.

To some extent the above heuristic is similar to the process described in Dror and Langevin [46] for transforming the directed Clustered Rural Postman Problem in a Generalized Traveling Salesman Problem.

The complexity of this heuristic is dominated by the complexity of the initial phase that builds the graphs $C_k^{i,j}$. Observe that once the pair i, j is fixed, finding $C_k^{i,j}$ requires $\mathcal{O}(n^3)$. Since the number of D-odd pairs is bounded by $\mathcal{O}(n^2)$, the overall complexity is $\mathcal{O}(n^5)$.

As an example, Figure 5.4 shows the merging clusters evolution on the instance graph P02 of the Christofides benchmark.

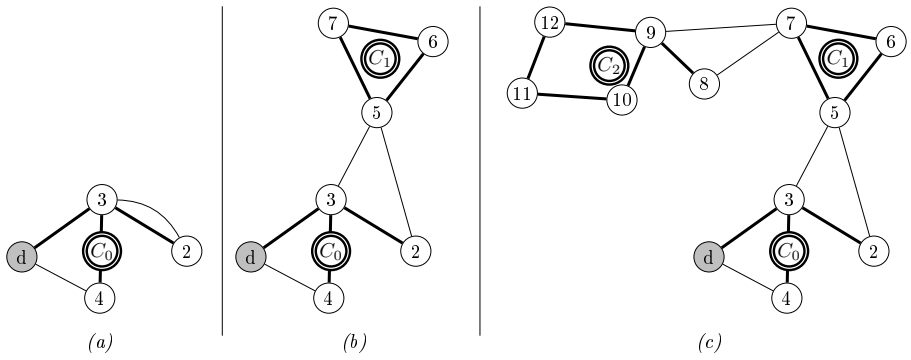

Figure 5.4: *Iterations made by the merging-cluster heuristic: (a) G^0. Initial solution equal to C_0^e. (b) G^1. Solution after the first merging (merge-edge-even). (c) G^2. An Optimal solution obtained after de second iteration (merge-vertex-odd) by merging clusters.*

5.1.2 Spanning tree heuristic

Next we present a heuristic algorithm based on spanning trees. The basic idea of this heuristic consists of finding a minimum cost spanning tree on a suitable subgraph and then obtaining the minimum cost Eulerian graph that contains the spanning tree. This idea was first proposed by Christofides [29] for the TSP and after by Frederickson for the RPP [57]. For the RPP the idea of Frederickson was further exploited by Fernández et al., [10], by taking into account in several ways the information given by the optimal solution to the LP relaxation. Next we present a heuristic that is the adaptation to the CPARP of the heuristic of Fernández et al. [53]. In particular, first we have to decide the clusters that are going to be serviced, since once the set of serviced clusters has been decided,

the problem reduces to a rural postman problem. Then, we apply the heuristic of [10].

Throughout, let $\widehat{K} \subseteq \{0, \ldots, p\}$ denote the set of clusters to be serviced. Let also $G_{\widehat{K}} = (V_{\widehat{K}}, E_{\widehat{K}})$ denote an auxiliary graph where $V_{\widehat{K}}$ contains one vertex associated with each component with index in the set $\widehat{K}$. Each pair of vertices $k_1, k_2 \in V_{\widehat{K}}$, $k_1 \neq k_2$, is connected by an edge.

The heuristic has two phases. In the first one we select the clusters that are going to be serviced as the ones of $\widehat{K}$, and we identify the minimum cost spanning tree T in $G_{\widehat{K}}$. The result of the first phase is the connected structure $T \cup \{C_k \mid k \in \widehat{K}\}$. Observe that some vertices may have odd degree in $T \cup \{C_k \mid k \in \widehat{K}\}$. Thus, in the second phase we find the minimum cost perfect matching M in the subgraph of K_n induced by the vertices with odd degree in $T \cup \{C_k \mid k \in \widehat{K}\}$.

The above scheme can be applied with different choices for the set $\widehat{K}$ and with different cost functions for building the minimum spanning tree. In our implementation, we exploit the information of the solution of the LP relaxation.

Given that the solution of the LP relaxation uses the best possible elements, the idea is to try to obtain a feasible solution that is as similar as possible to that fractional solution. Hereby, the heuristic solution will service all clusters that in the solution to the LP relaxation their associated variables have positive value.

Therefore, we consider $\widehat{K} = K(z^*) = \{k \in \{0, \ldots, p\} \mid z^*_k > 0\}$ where (x^*, y^*, z^*) denote the optimal solution to the LP relaxation.

Then, we repeat the above procedure three times, each of them with a different cost function for building the spanning tree in the first phase. These are the following:

For $g = \{v_{k_1}, v_{k_2}\} \in E_{z^*}$,

- $c^1_g = \min\{c_e : e = \{i, j\} \in H_2, i \in V_{k_1}, j \in V_{k_2}\}$.
- $c^2_g = \min\{1 - x^*_e : e = \{i, j\} \in H_2, i \in V_{k_1}, j \in V_{k_2}\}$.
- $c^3_g = \min\{(c_e)(1 - x^*_e) : e = \{i, j\} \in H_2, i \in V_{k_1}, j \in V_{k_2}\}$.

A scheme of the heuristic is given in Algorithm 5.2.

In general, in heuristic methods different policies for tie breaking may result in different solutions. This is also the case of the above heuristic, where most often we find ties for building the minimum cost spanning tree in the first phase.

1. Define $\widehat{K}$ and the auxiliary graph $G_{\widehat{K}} = (V_{\widehat{K}}, E_{\widehat{K}})$.
2. For $r = 1, 2, 3$
   - **1.1** Find a minimum spanning tree T^r in $G_{\widehat{K}}$ using function c^r.
   - **1.2** Identify the set of odd vertices $O^r \subseteq V$ in the graph $T^r \cup (\cup_{k \in \widehat{K}} C_k)$.
   - **2.1** Find a minimum cost perfect matching M^r, using the original cost function c in the subgraph of K_n induced by the vertex set O^r.
   - **2.2** $S^r = T^r \cup M^r$ is a feasible solution to CPARP.

Algorithm 5.2 *Spanning tree heuristic.*

Next we describe the policy for tie breaking that we have used, which depends on the cost function that is considered. A policy of tie breaking might be seen as a sequence of conditional expressions, in which the position of the condition in the sequence defines its priority. This means that a condition is checked only when all previous ones are true. In the above heuristic the following policies were used:

- Cost function c^1: Ties appear when two or more edges connecting the same pair of clusters have equivalent minimum cost. Then we break ties by selecting the edge which is incident with a smallest number of D-even vertices.
- Cost function c^2: Ties appear when two or more edges connecting the same pair of clusters have equivalent maximum solution values. Then, among edges connecting the same pair of clusters, the one that has minimum cost on the original data graph is chosen. If we keep on having ties, then we select the edge incident with a smallest number of D-even vertices.
- Cost function c^3: In this case we proceed as in the case of c^2.

5.2 Local Search

Next we describe two simple improvements that we apply to the feasible heuristics presented so far. In both cases we only do the move if it results in a feasible solution with a better objective function value than the original one. The two improvements have been called *shortcuts* and *interchanges*.

5.2.1 Shortcuts

Let $\mathcal{S}$ denote the current solution and let $P(u, v) \in \mathcal{S}$, be a path of edges which are traversed without profit, between vertices u and v. If $c_{u,v} < c(P)$ we say that edge uv is a shortcut for path P. Given a solution graph, any path without profit can be substituted by the corresponding edge when its cost is lower. In fact, this is just the result of repeated applications of the triangle inequality. While it is clear that the result of this operation will result in a solution that satisfies the degree constraints of the vertices, it is possible that the resulting solution be no longer connected. Therefore, in the context of CPARP there is an obvious restriction to the application of this idea, which comes from the necessity of connected solutions.

5.2.2 Interchanges

We consider possible interchanges of two edges $e_1 = u_1v_1$ and $e_2 = u_2v_2$ in the current solution by edges $f_1 = u_1v_2$ and $f_2 = u_2v_1$ not in the solution. This will be profitable when $c_{e_1} + c_{e_2} > c_{f_1} + c_{f_2}$. The vertices u_1 and u_2 must be D-odd and belong to the same component in order to keep the connectivity, and no restriction is imposed on the vertices v_1 and v_2. Observe that, if the original solution is connected, the result of this interchange will also be connected and the parity of the nodes will not change.

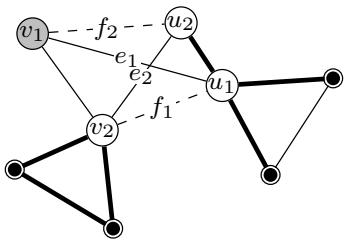

Figure 5.5: *An intermediate heuristic solution to some initial data graph.*

In Figure 5.5, we illustrate this idea graphically. If $(c_{e_1} + c_{e_2}) > (c_{f_1} + c_{f_2})$, then we would interchange edges e_1 and e_2, by f_1 and f_2.

The procedure is given in Algorithm 5.3. The function *other(vertex u,edge e)* returns the index of the neighbor of vertex u in edge e. This algorithm returns a logical value indicating whether the solution has been improved. Its running time is in $\mathcal{O}(n_o^4)$ being n_o the number of D_{odd} vertices in the solution.

```
bool INTERCHANGE_DODD()
{
    for_each cluster k in S
    {
        for_each D_odd vertex u_1 in V(C_k)
        {
            edge e_1 = (δ(u_1) ∩ S) \ D
            vertex v_1 = other(u_1, e_1)
            for_each D_odd vertex u_2 in V(C_k), u_2 ≠ u_1 and u_2 ≠ v_1
            {
                edge e_2 = (δ(u_2) ∩ S) \ D
                vertex v_2 = other(u_2, e_2)
                if (c_{e_1} + c_{e_2} > c_{u_1 v_2} + c_{u_2 v_1})
                {
                    S = S ∪ {e_{u_1 v_2}, e_{u_2 v_1}} \ {e_1, e_2}
                    return TRUE
                }
            }
        }
    }
    return FALSE
}
```

Algorithm 5.3 *Interchanges heuristic improvement.*

Chapter 6

Computational Results

In this chapter we present the results of the computational experiments that we have run with the algorithms presented in the previous chapters.

All the programs have been coded in C++. For the optimization of the linear problems, the routines of the CPLEX 9.0 callable library have been used. All the tests were run on an Intel CPU T2300 at 1.66 MHz, with 1GB.

Given that there were no available benchmark instances, we generated CPARP instances from the PRPP instances used in [10]. These, were in turn generated from a well known set of benchmark of RPP instances from [40, 69, 30]. For generating the PRPP instances, in [10], Aráoz, Fernández and Meza kept the cost function c and assigned random profits b to the edges. The random distribution used there depended on the nature of the edges:

- If e was a required edge for the RPP, then $b_e \sim U[c_e, 3c_e]$ in the PRPP.
- If e was a non-required for the RPP, then $b_e \sim U[0, c_e]$ in the PRPP.

In all cases vertex 1 was taken as the depot.

To transform the PRPP instances of [10] to instances for the CPARP, the set of demand edges has been defined as the set of required edges in the original RPP instances, and all profits assigned to non-demand edges have been set to zero.

Since the original 118 RPP instances are usually divided into five groups, we have also classified in five groups our CPARP instances, and we have named them like the original RPP instances. The group *ALBAIDA* contains two problems, *ALBAIDAA* and *ALBAIDAB* (see Corberán and Sanchis [40]). The group

CHRISTOFIDES contains 24 instances (labelled P) of Christofides et al. [30]. The last three groups contain instances from Hertz et al [69]: The *DEGREE* are 36 instances with vertices of degree 4 (labelled D), the *RANDOM* are 20 randomly generated instances (labelled R), and the *GRID* with 36 grid instances (labelled G).

This chapter is partitioned in three sections. First, the tables showing main parameters for the original data graphs are presented. After that, the results are listed and the last section presents the results of using the CPARP algorithm for solving the RPP.

6.1 Original Data Graphs

The main characteristics of the instances that we have used are listed in Table 6.1.

$\|V\|$	number of vertices in the original graph
$\|E\|$	number of edges in the original graph
$\|D\|$	number of demand edges
p	number of clusters
n	number of vertices in the transformed graph K_n
m	number of edges in the transformed graph K_n, $m = n(n-1)/2$
E_x	number of variables of type x
E_y	number of variables of type y

Table 6.1: *Column headings for tables related to the input data graphs.*

The number of clusters is not exactly the number of connected components of the graph induced by demand edges. When the depot cluster has no edges, it is counted as a cluster itself.

6.1.1 Albaida Instances

The first group contains two instances, *ALBAIDAA* and *ALBAIDAB*, obtained from a graph of the town of Albaida, Spain (see Corberán and Sanchis [40]). In Table 6.2 their size parameters are shown.

Note that in both cases, when the original graphs are transformed the number of vertices has not been reduced. Therefore, in the original instances all vertices have some demand edge in their cut. The costs of the edges of these instances range from 12 to 388, and the profits are between 17 and 1019.

name	$\|V\|$	$\|E\|$	$\|D\|$	p	n	m	E_x	E_y
ALBAIDAA	102	160	99	10	102	5151	4217	528
ALBAIDAB	90	144	88	11	90	4005	3672	1151

Table 6.2: *Main parameters of the ALBAIDA instances.*

6.1.2 Christofides Instances

The second group contains 24 instances, labelled P, of Christofides et al. [30]. Their sizes are listed in Table 6.3.

name	$\|V\|$	$\|E\|$	$\|D\|$	p	n	m	E_x	E_y
P01	11	13	7	4	11	55	53	4
P02	14	33	12	4	14	91	86	14
P03	28	57	26	4	28	378	323	114
P04	17	35	22	3	17	136	118	41
P05	20	35	16	5	20	190	177	32
P06	24	46	20	7	24	276	270	41
P07	23	47	24	3	23	253	226	27
P08	17	40	24	2	17	136	108	19
P09	14	26	14	3	14	91	81	13
P10	12	20	10	4	12	66	64	22
P11	9	14	7	3	9	36	34	7
P12	7	18	5	3	7	21	21	2
P13	7	10	4	3	7	21	21	2
P14	28	79	31	6	28	378	356	80
P15	26	37	19	8	26	325	321	31
P16	31	94	34	7	31	465	442	146
P17	19	44	17	5	19	171	165	22
P18	23	37	16	8	23	253	247	16
P19	33	54	29	7	33	528	512	45
P20	50	98	63	7	50	1225	1139	439
P21	49	110	67	6	49	1176	1079	358
P22	50	184	74	6	50	1225	1134	333
P23	50	158	78	6	50	1225	1133	385
P24	41	125	55	7	41	820	785	229

Table 6.3: *Main parameters of the CHRISTOFIDES instances.*

Note that, like in the *ALBAIDA* instances, in all cases $|V| = n$. This means that all original instances had all vertices with some demand edge in their cuts. Observe also that all these datasets correspond to small graphs (none has more than 50 vertices). The number of demand edges ranges from 4 to 78, and the number of clusters from 3 to 8. This results in instances with a number of x variables ranging from 21 to 1139, and a number of y variables between 2 and

439. All the costs of these instances are between 1 and 20, while the profits range from 1 to 40.

6.1.3 Hertz Degree Instances

This group is composed by 36 instances, labelled D. The vast majority, with vertices of degree 4. They are summarized in Table 6.4.

name	$\|V\|$	$\|E\|$	$\|D\|$	p	n	m	E_x	E_y
D0	16	32	3	2	5	10	10	1
D1	16	31	6	4	10	45	43	6
D2	16	31	9	4	12	66	64	12
D3	16	32	8	3	10	45	43	6
D4	16	31	8	5	12	66	64	11
D5	16	31	12	4	13	78	74	19
D6	16	32	11	5	14	91	89	14
D7	16	31	12	5	15	105	100	20
D8	16	31	16	4	16	120	113	27
D9	36	72	12	8	20	190	185	9
D10	36	72	10	8	18	153	153	10
D11	36	72	17	12	29	406	404	19
D12	36	72	17	8	25	300	291	34
D13	36	72	22	6	27	351	322	90
D14	36	72	30	9	35	595	579	63
D15	36	72	32	6	33	528	490	89
D16	36	72	34	5	35	595	528	117
D17	36	72	38	6	36	630	596	70
D18	64	128	28	15	43	903	895	35
D19	64	128	29	11	40	780	740	61
D20	64	128	27	12	38	703	684	51
D21	64	128	47	10	56	1540	1429	160
D22	64	128	47	12	57	1596	1503	151
D23	64	128	51	9	55	1485	1410	140
D24	64	128	68	6	61	1830	1615	251
D25	64	128	62	9	61	1830	1678	340
D26	64	128	75	5	62	1891	1663	364
D27	100	200	50	23	71	2485	2452	252
D28	100	200	55	20	71	2485	2441	254
D29	100	200	50	21	69	2346	2318	123
D30	100	200	86	14	89	3916	3669	970
D31	100	200	90	11	87	3741	3487	547
D32	100	200	81	16	88	3828	3657	588
D33	100	200	121	9	100	4950	4370	1301
D34	100	200	118	10	96	4560	4083	933
D35	100	200	116	9	96	4560	4113	817

Table 6.4: *Main parameters of the DEGREE instances.*

The number of vertices range from 16 to 100 while the edges from 31 to 200. The number of demand edges ranges from 3 to 121, and that of the clusters from 2 to 21. This factor has a great impact on the cpu time of the algorithms. All these values lead the number of x variables to range from 10 to 4370, and of y variables from 1 to 1301. All the costs are between 1 and 121, and the profits between 1 and 260.

6.1.4 Hertz Random Instances

We also run the algorithm over 20 randomly generated instances, labelled R. All of them are small graphs. Their main characteristics are shown in Table 6.5.

name	$\|V\|$	$\|E\|$	$\|D\|$	p	n	m	E_x	E_y
R0	20	37	3	4	7	21	21	3
R1	20	47	4	5	9	36	36	4
R2	20	47	4	4	8	28	28	3
R3	20	75	7	4	11	55	55	5
R4	20	60	6	4	10	45	45	6
R5	30	70	7	5	12	66	66	7
R6	30	112	10	5	15	105	101	9
R7	30	70	7	5	12	66	64	4
R8	30	111	11	6	17	136	132	13
R9	30	111	11	7	18	153	148	15
R10	40	130	13	9	22	231	231	13
R11	40	103	10	6	16	120	118	7
R12	40	82	8	6	14	91	89	7
R13	40	203	18	9	25	300	291	20
R14	40	203	18	8	25	300	281	19
R15	50	203	20	9	28	378	365	31
R16	50	162	15	12	27	351	349	12
R17	50	130	13	6	19	171	160	12
R18	50	203	19	7	26	325	302	26
R19	50	203	19	8	27	351	336	34

Table 6.5: *Main parameters of the RANDOM instances.*

As can be seen, the sizes range from 20 nodes and 37 edges to 50 nodes and 203 edges. The number of demand edges goes from 3 to 20 and the number of clusters from 4 to 12. This results in transformed graphs with a number of nodes ranging from 7 to 28. In these instances the number of x variables ranges from 21 to 365, whereas the number of y variables goes from 3 to 34. The *RANDOM* instances have the largest variability in the cost and profits among all on which we have worked on. All the costs are between 143 and 4284, while the profits range from 213 to 12579.

6.1.5 Hertz Grid Instances

The last group contains 36 instances corresponding to grid graphs, labelled G. They are summarized in Table 6.6.

.

name	$\|V\|$	$\|E\|$	$\|D\|$	p	n	m	E_x	E_y
G0	16	24	3	4	7	21	21	3
G1	16	24	5	6	11	55	55	5
G2	16	24	4	5	9	36	36	4
G3	16	24	8	5	13	78	78	5
G4	16	24	7	6	13	78	78	6
G5	16	24	7	4	10	45	43	10
G6	16	24	13	4	16	120	112	36
G7	16	24	8	5	13	78	76	10
G8	16	24	9	5	13	78	76	16
G9	36	60	11	8	19	171	166	12
G10	36	60	13	10	23	253	253	15
G11	36	60	15	9	24	276	270	13
G12	36	60	26	8	34	561	523	83
G13	36	60	23	6	27	351	334	25
G14	36	60	25	7	31	465	439	30
G15	36	60	35	5	34	561	503	101
G16	36	60	30	7	33	528	491	67
G17	36	60	34	5	35	595	537	93
G18	64	112	24	11	35	595	584	43
G19	64	112	27	13	40	780	760	86
G20	64	112	27	15	42	861	848	70
G21	64	112	46	9	50	1225	1115	238
G22	64	112	47	9	56	1540	1409	328
G23	64	112	50	12	59	1711	1614	162
G24	64	112	68	4	62	1891	1605	386
G25	64	112	61	6	61	1830	1621	307
G26	64	112	66	7	62	1891	1637	520
G27	100	180	41	19	60	1770	1755	80
G28	100	180	49	21	69	2346	2276	324
G29	100	180	44	20	64	2016	1988	157
G30	100	180	73	13	82	3321	3112	383
G31	100	180	77	18	92	4186	3956	630
G32	100	180	82	11	91	4095	3706	563
G33	100	180	113	4	97	4656	4000	602
G34	100	180	107	9	100	4950	4394	710
G35	100	180	109	6	97	4656	3991	790

Table 6.6: *Main parameters of the GRID instances.*

As can be seen, the sizes range from 16 nodes and 24 edges to 100 nodes and 180 edges. The number of demand edges goes from 3 to 113 and the number of clusters from 4 to 21. This results in transformed graphs with a number

of nodes ranging from 7 to 100. In these instances the number of x variables ranges from 21 to 4394, whereas the number of y variables goes from 3 to 790. As opposite to the previous group, in all these instances the costs are all equal to 1, and their profits range from 1 to 3.

6.2 Results

In this section we present the obtained results. First, the values related to the optimal solution of the LP relaxation (and so, related to the root node of the exploration tree) are listed in one table for each set of data graphs. After that, one more table is also presented with the values related to the exact algorithm only for those instances that have not been optimality solved at the root node.

6.2.1 Results of the LP Relaxation

The solutions to the LP relaxation give us upper bounds for the exploration phase, when no optimal solution is already found. In Table 6.7 the columns related to the results of the optimal solutions of the LP relaxations are listed.

Z_r	linear relaxation optimal value
t_{Z_r}	required CPU time for the linear relaxation solution (in seconds)
Z_0	linear relaxation first iteration solution value
iter	number of LP problems solved
con $\#(\leq)$	connectivity iterations (connectivity inequalities added)
coc $\#(\leq)$	cocircuit iterations (cocircuit inequalities added)
	instances with integer solution to the LP relaxation are marked with a ✓

Table 6.7: *Column headings for tables related to the root node of the exploration tree.*

Since all instances marked with ✓ in Tables 6.8 - 6.12 are optimally solved at the root node, they will not appear in the final tables. Note also that in this way we can compare the improvements between the value of the upper bound from the initial solution, Z_0, and the one given at the end of the iteration algorithm, Z_r.

For one of the *ALBAIDA* instances, we have already an integer solution to the linear relaxation problem. The problem given by *ALBAIDAA* is already optimally solved and therefore no more data will be given about it. The second one, *ALBAIDAB*, has not been optimally solved yet, and thus it requires to start the exploration. The values for both instances are given in Table 6.8.

name	Z_r	t_{Z_r}	Z_0	iter	con($\leq$)	coc($\leq$)	
ALBAIDAA	5179.00	5.890	5595.00	11	2(15)	8(23)	✓
ALBAIDAB	3394.29	8.672	3870.00	18	8(47)	9(34)	

Table 6.8: *LP solutions of the ALBAIDA instances.*

As can be observed in Table 6.8, for instances with more than 100 vertices and 10 or 11 clusters, we have both solution times less than 10 seconds. Note also that the number of global iterations is reasonably small. This number is somehow related to the improvement done by the separation phases. In fact, in both cases, the value of the solution of the initial model exceeds to that of the LP relaxation in less than a 15%.

The values for the *CHISTOFIDES* instances are given in Table 6.9.

name	Z_r	t_{Z_r}	Z_0	iter	con($\leq$)	coc($\leq$)	
P01	3.00	0.000	3.00	1	0(0)	0(0)	✓
P02	28.00	0.047	49.00	4	2(5)	1(4)	
P03	47.00	0.281	52.00	5	2(5)	2(10)	✓
P04	33.00	0.047	37.00	3	1(2)	1(4)	
P05	19.00	0.031	26.00	2	1(3)	0(0)	✓
P06	47.00	0.063	48.00	2	0(0)	1(4)	✓
P07	73.00	0.062	76.00	2	1(2)	0(0)	✓
P08	72.00	0.032	73.00	2	0(0)	1(2)	✓
P09	38.00	0.015	42.50	4	1(2)	2(6)	✓
P10	35.00	0.015	43.00	3	1(2)	1(4)	✓
P11	9.00	0.000	9.00	1	0(0)	0(0)	✓
P12	8.00	0.000	10.00	2	1(2)	0(0)	✓
P13	3.00	0.016	3.00	1	0(0)	0(0)	✓
P14	98.17	0.234	106.00	9	5(16)	3(12)	
P15	15.00	0.063	21.00	3	2(5)	0(0)	✓
P16	73.00	0.297	83.00	7	3(10)	3(10)	
P17	22.00	0.047	29.00	5	2(5)	2(7)	
P18	0.80	0.125	22.00	11	8(28)	2(6)	✓
P19	49.00	0.157	52.00	4	2(7)	1(3)	✓
P20	189.67	1.140	211.00	10	4(15)	5(16)	
P21	221.00	0.781	231.00	8	4(14)	3(9)	
P22	398.00	0.797	405.00	7	3(12)	3(9)	
P23	298.00	0.656	307.50	6	4(14)	1(1)	
P24	173.00	0.375	184.00	4	2(11)	1(5)	

Table 6.9: *LP solutions of the CHRISTOFIDES instances.*

There are 14 out of 24 optimally solved datasets that do not resort to exploration tree, i.e. 13 optimal solutions of the LP relaxation that are integer

and P18, solved optimally with empty solution. Column t_{Z_r} of Table 6.9 shows that all the instances required less than one second of the cpu time but P20, that required slightly more. The values obtained with the initial model, Z_0, are reasonably close to the final, Z_r. This leads to give such small number of iterations (all but two cases are smaller than 10 iterations).

With respect to the *DEGREE* instances, the values are depicted in Table 6.10. Here, the number of integer solutions rises up to 27 out of 36.

name	Z_r	t_{Z_r}	Z_0	iter	con($\leq$)	coc($\leq$)	
D0	109.00	0.000	109.00	1	0(0)	0(0)	✓
D1	0.00	0.015	37.00	2	1(2)	0(0)	✓
D2	123.00	0.031	123.00	1	0(0)	0(0)	✓
D3	109.00	0.031	109.00	1	0(0)	0(0)	✓
D4	35.00	0.000	35.00	1	0(0)	0(0)	✓
D5	270.00	0.016	270.00	1	0(0)	0(0)	✓
D6	115.00	0.015	149.00	2	1(4)	0(0)	✓
D7	55.00	0.032	175.00	4	3(8)	0(0)	✓
D8	367.00	0.015	387.50	2	1(3)	0(0)	✓
D9	80.00	0.047	107.50	3	2(4)	0(0)	✓
D10	0.00	0.078	5.00	2	1(2)	0(0)	✓
D11	136.00	0.235	225.50	6	5(36)	0(0)	✓
D12	103.00	0.094	141.50	4	2(5)	1(3)	✓
D13	496.00	0.313	580.25	12	4(12)	7(18)	✓
D14	545.00	0.328	594.00	6	3(11)	2(11)	✓
D15	528.50	0.203	543.50	5	2(4)	2(6)	
D16	801.00	0.328	856.50	7	2(6)	4(13)	
D17	871.00	0.234	909.50	3	0(0)	2(7)	✓
D18	256.00	0.844	362.00	9	6(42)	2(6)	✓
D19	282.00	0.250	293.00	4	2(9)	1(3)	✓
D20	180.00	0.250	262.00	3	2(15)	0(0)	✓
D21	768.00	1.266	810.50	10	7(27)	2(8)	✓
D22	733.00	2.000	801.00	13	8(59)	4(16)	✓
D23	738.00	1.109	832.00	10	6(21)	3(12)	
D24	1281.00	0.516	1281.00	1	0(0)	0(0)	✓
D25	1012.60	1.765	1045.00	9	2(11)	6(18)	
D26	1310.00	0.797	1400.50	4	2(4)	1(6)	✓
D27	241.00	37.797	492.00	67	64(778)	2(16)	✓
D28	483.00	5.828	597.50	21	18(209)	2(8)	✓
D29	175.00	7.734	300.25	29	27(309)	1(7)	✓
D30	933.00	9.485	1190.00	26	23(155)	2(15)	
D31	1195.80	5.719	1287.00	14	8(48)	5(26)	
D32	720.83	9.235	780.50	15	7(37)	7(22)	
D33	1567.50	5.781	1669.83	10	6(26)	3(21)	
D34	1444.00	9.812	1561.00	18	10(54)	7(23)	
D35	1579.00	5.922	1647.50	12	2(15)	9(24)	✓

Table 6.10: *LP solutions of the DEGREE instances.*

As can be observed in Table 6.10, in general the number of iterations is

small, and also the cpu times required are very satisfactory in almost all instances. Instance D27 required more than half a minute to be solved, needing 67 iterations. All but this case required less than 30 iterations and there is a considerable number of instances that required less than 10 iterations.

The optimal solution values of the LP relaxations of the *RANDOM* instances can be seen in Table 6.11. The number of integer solutions in this group is 19

name	Z_r	t_{Z_r}	Z_0	iter	con($\leq$)	coc($\leq$)	
R0	0.00	0.016	0.00	1	0(0)	0(0)	✓
R1	236.00	0.015	2982.00	3	2(7)	0(0)	✓
R2	0.00	0.000	4503.00	2	1(2)	0(0)	✓
R3	8605.00	0.016	9545.50	2	1(2)	0(0)	✓
R4	11027.00	0.016	13412.50	3	1(3)	1(3)	✓
R5	0.00	0.016	2772.00	3	2(4)	0(0)	✓
R6	6681.00	0.062	9566.00	3	2(6)	0(0)	✓
R7	3556.00	0.016	5304.00	2	1(3)	0(0)	✓
R8	1132.00	0.063	11057.00	6	4(12)	1(2)	✓
R9	7079.00	0.078	11276.00	6	2(7)	3(8)	✓
R10	6025.00	0.219	11971.00	10	9(46)	0(0)	✓
R11	2603.00	0.031	5086.50	2	1(2)	0(0)	✓
R12	0.00	0.016	8819.00	3	2(5)	0(0)	✓
R13	18059.00	0.250	22443.00	10	5(26)	4(12)	✓
R14	17023.75	0.250	18914.00	8	4(15)	3(6)	
R15	10794.00	0.375	18289.50	13	7(24)	5(8)	✓
R16	4544.00	0.078	6471.00	3	2(9)	0(0)	✓
R17	1732.00	0.063	8403.50	3	2(6)	0(0)	✓
R18	14055.00	0.125	22292.50	5	4(14)	0(0)	✓
R19	17958.00	0.172	18683.00	3	1(2)	1(4)	✓

Table 6.11: *LP solutions of the RANDOM instances.*

out of 20. Only R14 needs to resort to the exploration algorithm. The solution values of the initial model in Table 6.11 are really far away from those of the final LP relaxation. However, it is due to the large variability on the costs and profits of the edges of these instances, since the number of iterations is small (less than 15 in all cases). Thus, it seems clear that the distance between the value of the solution of the initial model and the optimal to the LP relaxation is strongly related to the distribution of the costs and profits of the edges.

Finally, in Table 6.12 the results of the *GRID* instances are listed. Here we have 21 of 36 integer solutions. In Table 6.12 we can see again that the required time for the LP relaxation algorithm is small. The instance that takes more time is G31, that needs more than 25 seconds, but all the others required less than 12 seconds.

name	Z_r	t_{Z_r}	Z_0	iter	con($\leq$)	coc($\leq$)	
G0	0.00	0.000	0.00	1	0(0)	0(0)	✓
G1	0.00	0.015	1.00	4	3(7)	0(0)	✓
G2	0.00	0.016	0.50	2	1(2)	0(0)	✓
G3	2.00	0.047	2.00	1	0(0)	0(0)	✓
G4	0.00	0.031	2.00	6	4(14)	1(2)	✓
G5	4.00	0.016	5.00	3	2(5)	0(0)	✓
G6	7.00	0.031	9.00	4	2(5)	1(3)	
G7	1.00	0.016	3.00	4	3(9)	0(0)	✓
G8	4.00	0.015	5.00	3	2(6)	0(0)	✓
G9	1.00	0.079	4.50	8	6(26)	1(3)	
G10	0.00	0.141	2.00	7	6(31)	0(0)	✓
G11	3.00	0.093	5.00	4	3(17)	0(0)	✓
G12	13.00	0.438	16.00	9	4(19)	4(12)	✓
G13	8.50	0.172	13.00	6	4(13)	1(7)	
G14	13.00	0.219	15.00	7	5(19)	1(4)	✓
G15	23.00	0.156	24.00	2	0(0)	1(3)	✓
G16	15.50	0.312	19.00	7	5(19)	1(10)	
G17	20.00	0.203	20.00	4	2(6)	1(3)	✓
G18	6.00	0.375	8.50	7	5(39)	1(3)	
G19	6.00	0.954	11.00	15	13(99)	1(6)	
G20	9.00	0.813	12.00	14	13(114)	0(0)	✓
G21	29.00	1.125	32.50	11	6(33)	4(15)	✓
G22	30.00	1.015	33.00	7	4(20)	2(9)	✓
G23	29.00	1.625	29.00	8	5(39)	2(10)	✓
G24	50.00	1.578	51.00	10	2(5)	7(25)	✓
G25	39.00	1.438	42.00	9	4(14)	4(12)	
G26	51.00	1.109	54.00	6	3(11)	2(9)	✓
G27	13.00	3.906	16.00	19	15(201)	3(10)	
G28	22.57	11.766	27.00	32	28(289)	3(15)	
G29	12.33	5.969	18.00	23	17(154)	5(14)	
G30	41.70	10.187	44.50	23	16(114)	6(24)	
G31	48.00	26.016	51.00	37	32(353)	4(23)	
G32	49.33	5.031	51.00	12	6(25)	5(20)	
G33	80.00	2.329	82.00	3	0(0)	2(6)	✓
G34	76.00	6.453	77.00	13	7(30)	5(21)	
G35	78.00	5.312	80.00	11	3(12)	7(21)	

Table 6.12: *LP solutions of the GRID instances.*

In global, we added more connectivity cuts than cocircuit cuts. In 615 iterations we added 4293 connectivity cuts, whereas in 544 iterations we added 1392 cocircuit cuts. This is, a 76% of the total inequalities added in the iterative LP solver scheme are of connectivity. However, the role of the cocircuit inequalities is very important and we have observed that they contribute substantially to the improvement of the upper bounds for the exploration.

6.2.2 Results at the Root Node of the Exact Algorithm

Before starting the branch and cut algorithm the heuristic algorithm explained in Chapter 5 is applied. The heuristic value of the solution helps pruning the exploration tree, or even better, it gives directly an optimal solution when the difference to the LP optimal solution is less than 1.

Next we analyze the results of the heuristic algorithm. In particular, we observe the quality of the solution obtained with each of the considered cost functions explained in Subsection 5.1.2 of page 74.

name	Z_h	t_{Z_h}	$Z(c^1)$	$Z(c^2)$	$Z(c^3)$	Z_r	
ALBAIDAB	3388	0.515	3388	3308	3388	3394.29	
P02	26	0.000	26	26	26	28.00	
P04	31	0.016	31	29	31	33.00	
P14	89	0.078	89	81	89	98.17	
P16	71	0.078	69	71	71	73.00	
P17	19	0.016	19	19	19	22.00	
P20	180	0.250	180	178	178	189.67	
P21	208	0.219	208	202	208	221.00	
P22	395	0.188	394	395	393	398.00	
P23	295	0.235	295	292	295	298.00	
P24	171	0.109	171	171	171	173.00	
D15	520	0.078	520	520	520	528.50	
D16	758	0.078	752	758	749	801.00	
D23	724	0.250	724	724	724	738.00	
D25	991	0.250	983	973	991	1012.60	
D30	870	0.531	870	798	830	933.00	
D31	1162	0.563	1159	1162	1159	1195.80	
D32	656	0.609	647	656	647	720.83	
D33	1550	0.687	1550	1530	1550	1567.50	
D34	1430	0.782	1405	1387	1430	1444.00	
R14	15730	0.047	15730	15648	14665	17023.75	
G6	6	0.015	6	4	4	7.00	
G9	1	0.000	1	1	1	1.00	✓
G13	8	0.047	8	8	8	8.50	✓
G16	15	0.062	15	15	15	15.50	✓
G18	6	0.079	6	6	6	6.00	✓
G19	5	0.078	5	5	5	6.00	
G25	39	0.172	37	39	39	39.00	✓
G27	9	0.250	5	9	5	13.00	
G28	17	0.438	13	17	17	22.57	
G29	9	0.422	5	9	7	12.33	
G30	40	0.344	36	38	40	41.70	
G31	44	0.703	40	44	40	48.00	
G32	48	0.531	48	46	48	49.33	
G34	76	0.516	74	76	76	76.00	✓
G35	78	0.484	76	78	78	78.00	✓

Table 6.13: *Heuristics solution values for no optimally solved graphs.*

The results are given in Table 6.13. Only instances that were not optimally solved with the iterative LP solver scheme are considered. In Table 6.13, Column Z_h is the value of the heuristic solution and t_{Z_h} is the cpu time required for obtaining the heuristic. As could be expected, the cpu times required for the heuristic algorithm once the LP relaxation has been optimally solved is small (less than one second) for all the instances, since it is polynomial on p. The next three columns, $Z(c^i)$, $i \in \{1, 2, 3\}$, are the values of the heuristic solution calculated with the mentioned costs. Therefore, the column Z_h corresponds to the $max\{Z(c^i)\}$, for $i = 1, 2, 3$. Usually c^1 gives the best results, that is the minimum cost in the original data graph between each pair of clusters in the LP optimal solution. This criterion seems to be correlated with the c^3, since in many instances where the c^1 gives the best result, $c^3 = c^1$. In other cases, however, the situation is somehow inverted, and the best result is obtained by the complementary of the solution value, c^2. Finally, in Table 6.13 is also shown the value of Z_r, the optimal value of the LP relaxation already seen in the tables above.

6.2.3 Results of the Exploration

From the results of the previous tables we can see that after solving the LP relaxation at the root node, the optimality of the obtained solutions could be proved for 80 instances. In addition, with the values listed in Table 6.13, the upper bound allowed us to deduce that the heuristic solution was optimal for 9 more instances. That is, a total of 89 out of the 118 considered instances were optimally solved at the root node, what means a 75.4%.

Therefore, the number of the remaining graphs not optimally solved yet raises up to 29 instances. Thus, we run the exact algorithm branching on the variables explained in Subsection 4.3. Their results follow.

In Table 6.14 Column Z^* is the optimal solution of the CPARP instance. Next, below the heading of t_{exp}, the cpu time used at the exploration time is indicated. In the third column, with the heading $nodes(Z + X)$, we have the number of explored nodes in the search tree. This is, the total number of nodes explored, and between parenthesis this number is separated on the number of nodes generated when branching on a z variable, and the number of nodes generated when branching on a x variable. Observe that most instances are solved at stage 1 of the tree depicted in Figure 4.10 of Subsection 4.3. Last column corresponds to the total required cpu time for computing this solution. Note that the required time is mainly used in the exploration tree.

As can be seen, in general the number of nodes in the exploration is small. We can observe two pathological cases, P20, and specially, D30, whose required cpu time goes up until almost one hour. For the remaining instances, the exploration reduces to a few nodes.

name	Z^*	t_{exp}	$nodes(Z+X)$	t_{Z^*}
ALBAIDAB	3388	1.656	1 (1 + 0)	10.609
P02	26	0.032	3 (0 + 3)	0.063
P04	31	0.094	4 (0 + 4)	0.141
P14	92	4.062	50 (1 + 49)	4.390
P16	72	0.297	2 (1 + 1)	0.578
P17	20	0.172	6 (1 + 5)	0.234
P20	182	97.391	160 (14 + 146)	98.578
P21	214	20.453	43 (0 + 43)	21.281
P22	395	0.781	2 (0 + 2)	1.594
P23	295	0.562	2 (0 + 2)	1.250
P24	171	2.891	19 (0 + 19)	3.297
D15	520	0.125	1 (1 + 0)	0.313
D16	783	1.672	10 (0 + 10)	2.031
D23	724	0.625	2 (2 + 0)	1.797
D25	1011	0.890	2 (1 + 1)	2.734
D30	883	3223.453	830 (61 + 769)	3233.078
D31	1195	1.297	1 (1 + 0)	7.454
D32	720	2.672	1 (1 + 0)	12.063
D33	1550	33.813	19 (2 + 17)	39.688
D34	1436	26.250	8 (2 + 6)	36.375
R14	16944	0.110	1 (1 + 0)	0.360
G6	6	0.032	1 (1 + 0)	0.079
G19	5	1.218	1 (1 + 0)	2.203
G27	13	2.125	1 (1 + 0)	6.157
G28	21	18.672	6 (6 + 0)	30.703
G29	12	1.578	1 (1 + 0)	7.797
G30	41	5.782	2 (2 + 0)	16.235
G31	48	70.703	7 (6 + 1)	97.922
G32	49	34.890	17 (2 + 15)	40.343

Table 6.14: *Results of the exploration algorithm instances.*

Summarizing, we can conclude that the results of our computational experiment are, in general, very good. More than 75% instances were optimally solved at the root node. The cpu times (excepting for one instance) were smaller than 10 seconds. The remaining 29 instances were optimally solved in the enumeration tree in times that, with the exception of three instances, were smaller than 100 seconds.

6.3 The CPARP algorithm for the RPP

Even though the object of this thesis is to study the CPARP, in the computational experiments we have also used our CPARP formulation and exact algorithm to solve the whole set of RPP instances. Two main reasons lead to this decision. On the one hand, because in Subsection 3.1 we gave a polynomial reduction for transforming Rural Postman Problem to CPARP instances. On the other hand, because we also wanted to test the quality of our CPARP algorithm by comparing it with other RPP algorithms in the literature.

Since the considered instances had originally been RPP instances, now the set of required edges are those of the original instances which, as previously explained, always coincide with the sets of demand edges in the CPARP instances. Also, the type of the problem has been changed, since by definition of the RPP, now we are facing minimization problems. And, finally, the objective function is slightly different, since it does not take into account profits at all. In general, we will see that the good behavior of the algorithm for the CPARP maintains its quality when solving RPP instances.

The polynomial reduction has been carried out exactly as explained in Subsection 3.1, so profits for demand edges in the CPARP instances have been raised up to one million. Therefore, all required edges of the RPP instances are traversed in the optimal solutions next presented.

In Tables 6.15 - 6.19 the results are given. All the tables have eight columns for each instance with its RPP optimal value (z^*), the value of the optimal solution to the relaxation of the formulation described in 3.4.4 (z_r), the heuristic value at the root node of the exploration tree (z_h), the percent gap between the optimal solution and the solution to the relaxed model, $g_r = 100(z^* - z_r)/z_r$, the percent gap between the heuristic solution value and the optimal one, $g_h = 100(z_h - z^*)/z^*$, the number of nodes visited in the exploration tree (n), and the cpu times, in seconds, required to solve the relaxed LP formulation (t_{z_r}) and to optimally solve the problem (t_{z^*}).

In general, note that given an instance, if $t_{z^*} = t_{z_r}$ then it has been solved at the root node of the exploration tree since its LP relaxation already gave an optimal solution. Therefore, when in a row both times are the same, then the number of visited nodes in the exploration tree is zero. However, among the instances with $t_{z^*} > t_{z_r}$, there are also some that did not require exploring any additional node. In particular, those for which optimally of their heuristic solution was proven at the root node.

As can be seen in Table 6.15, in both *ALBAIDA* instances the required time to achieve an integer solution is small. Only for the instances of CPARP whose optimal solution services all the clusters the number of nodes in the exploration tree coincide. This is not the case of *ALBAIDAA* that was optimally solved at the root node of the exploration tree in the case of the CPARP, and as a RPP

name	z^*	z_r	z_h	g_r	g_h	n	t_{z_r}	t_{z^*}
ALBAIDAA	10599	10592.67	10599	0.1	0.0	5	20.140	24.969
ALBAIDAB	8629	8623.67	8629	0.1	0.0	1	9.625	11.781

Table 6.15: *Solving the RPP on ALBAIDA instances with the CPARP algorithm.*

it requires to explore five nodes.

name	z^*	z_r	z_h	g_r	g_h	n	t_{z_r}	t_{z^*}
P01	76	72.00	76	5.6	0.0	0	0.031	0.031
P02	152	142.50	153	6.7	0.7	5	0.047	0.188
P03	101	101.00	101	0.0	0.0	0	0.234	0.234
P04	84	82.00	84	2.4	0.0	4	0.047	0.141
P05	124	118.50	124	4.6	0.0	0	0.062	0.109
P06	102	102.00	102	0.0	0.0	0	0.110	0.110
P07	130	130.00	130	0.0	0.0	0	0.062	0.062
P08	122	117.00	122	4.3	0.0	0	0.047	0.047
P09	83	80.00	83	3.7	0.0	0	0.031	0.031
P10	80	78.00	80	2.6	0.0	1	0.032	0.063
P11	23	23.00	23	0.0	0.0	0	0.015	0.015
P12	19	19.00	19	0.0	0.0	0	0.000	0.000
P13	35	35.00	35	0.0	0.0	0	0.016	0.016
P14	202	192.50	205	4.9	1.5	30	0.219	2.984
P15	441	436.00	441	1.1	0.0	0	0.235	0.235
P16	203	188.00	203	8.0	0.0	9	0.281	1.140
P17	112	110.50	112	1.4	0.0	0	0.110	0.141
P18	146	144.00	146	1.4	0.0	0	0.141	0.141
P19	257	253.00	257	1.6	0.0	0	0.218	0.218
P20	398	386.33	400	3.0	0.5	124	1.360	80.641
P21	366	344.00	372	6.4	1.6	40	0.828	18.235
P22	621	614.00	622	1.1	0.2	3	0.891	1.860
P23	475	437.00	475	8.7	0.0	2	0.703	1.297
P24	405	399.00	405	1.5	0.0	12	0.453	2.422

Table 6.16: *Solving the RPP on CHRISTOFIDES instances with the CPARP algorithm.*

The results corresponding to the *CHRISTOFIDES* instances are displayed in Table 6.16. Observe that exactly 14 instances were solved at the root node of the exploration tree, and for the rest of them, the values of the gaps are under the 9%. In our opinion the largest percent gaps could be reduced if, instead of using the heuristic separation for the cocircuit inequalities in the general case of Subsection 4.2.2 we used the exact algorithm described in Algorithm 4.4. Also noticeable are the required cpu times for optimally solving all the instances: all but one under one second.

name	z^*	z_r	z_h	g_r	g_h	n	t_{z_r}	t_{z^*}
D0	272	272.00	272	0.0	0.0	0	0.016	3.891
D1	701	701.00	701	0.0	0.0	0	0.032	0.032
D2	702	702.00	702	0.0	0.0	0	0.032	0.032
D3	754	753.00	754	0.1	0.0	0	0.031	0.047
D4	920	860.00	920	7.0	0.0	3	0.031	0.094
D5	963	921.83	963	4.5	0.0	8	0.031	0.187
D6	909	908.00	909	0.1	0.0	0	0.063	0.063
D7	1026	937.50	1039	9.4	1.3	7	0.046	0.234
D8	1013	989.50	1013	2.4	0.0	3	0.062	0.172
D9	1032	1031.00	1032	0.1	0.0	0	0.172	0.172
D10	739	739.00	739	0.0	0.0	0	0.203	0.203
D11	1088	1088.00	1088	0.0	0.0	0	0.219	0.219
D12	1085	1083.00	1085	0.2	0.0	0	0.156	0.172
D13	1097	1097.00	1097	0.0	0.0	0	0.390	0.390
D14	1416	1416.00	1416	0.0	0.0	0	0.360	0.360
D15	1405	1402.00	1405	0.2	0.0	0	0.328	0.328
D16	1436	1418.00	1449	1.3	0.9	10	0.437	1.844
D17	1683	1683.00	1683	0.0	0.0	0	0.312	0.328
D18	1264	1261.00	1264	0.2	0.0	0	1.437	1.453
D19	1373	1373.00	1373	0.0	0.0	0	1.047	1.047
D20	1262	1261.00	1262	0.1	0.0	0	0.703	0.703
D21	1602	1599.00	1602	0.2	0.0	0	1.140	1.156
D22	1739	1720.00	1739	1.1	0.0	8	3.578	10.890
D23	1774	1769.00	1774	0.3	0.0	0	1.625	1.625
D24	2188	2185.00	2188	0.1	0.0	0	1.328	1.344
D25	1989	1983.00	2014	0.3	1.3	1	1.921	2.843
D26	2441	2436.00	2441	0.2	0.0	0	1.281	1.297
D27	1793	1790.82	1837	0.1	2.5	1	30.188	33.797
D28	1834	1833.00	1842	0.1	0.4	2	10.485	13.500
D29	1766	1734.94	1808	1.8	2.4	82	11.062	236.203
D30	2310	2298.38	2310	0.5	0.0	5	10.593	21.468
D31	2539	2531.17	2551	0.3	0.5	11	17.219	35.625
D32	2137	2133.00	2137	0.2	0.0	0	14.516	14.547
D33	2914	2872.67	2931	1.4	0.6	158	8.094	436.844
D34	2906	2902.00	2906	0.1	0.0	0	7.141	7.172
D35	2820	2815.00	2820	0.2	0.0	0	6.187	6.218

Table 6.17: *Solving the RPP on DEGREE instances with the CPARP algorithm.*

For the *DEGREE* instances, in Table 6.17, the results are also satisfactory, since 23 out of 36 instances were optimally solved at the root node of the exploration tree, and the relaxation gap (in all but two instances) is better than in the previous case (under 5%). Almost in all of them the time required was less than 40 seconds. However, we have two pathological cases, D29 and D33, whose times raised up to almost four and eight minutes respectively. Nevertheless, these two instances present more reasonable times for solving only the LP relaxation, and as also can be seen in Table 6.17, their relaxation gaps, g_r, are very good (under 2%).

The solution values for the *RANDOM* instances are shown in Table 6.18.

name	z^*	z_r	z_h	g_r	g_h	n	t_{z_r}	t_{z^*}
R0	29853	29853.00	29853	0.0	0.0	0	0.047	0.047
R1	34317	34317.00	34317	0.0	0.0	0	0.031	0.031
R2	24968	24968.00	24968	0.0	0.0	0	0.032	0.032
R3	34054	34054.00	34054	0.0	0.0	0	0.031	0.031
R4	26256	26256.00	26256	0.0	0.0	0	0.031	0.031
R5	35758	35758.00	35758	0.0	0.0	0	0.078	0.078
R6	39221	39221.00	39221	0.0	0.0	0	0.094	0.094
R7	35140	35140.00	35140	0.0	0.0	0	0.062	0.062
R8	48729	48729.00	48729	0.0	0.0	0	0.109	0.109
R9	42572	42572.00	42572	0.0	0.0	0	0.125	0.125
R10	38741	38741.00	38741	0.0	0.0	0	0.344	0.360
R11	42628	42628.00	42628	0.0	0.0	0	0.296	0.296
R12	50586	50586.00	50586	0.0	0.0	0	0.157	0.157
R13	54867	54867.00	54867	0.0	0.0	0	0.281	0.281
R14	65549	65549.00	65549	0.0	0.0	0	0.312	0.312
R15	63714	63714.00	63714	0.0	0.0	0	0.672	0.610
R16	50349	50349.00	50349	0.0	0.0	0	0.437	0.437
R17	43272	43272.00	43272	0.0	0.0	0	0.282	0.282
R18	56870	56870.00	56870	0.0	0.0	0	0.343	0.343
R19	45331	45331.00	45331	0.0	0.0	0	0.391	0.391

Table 6.18: *Solving RPP on RANDOM instances with the CPARP algorithm.*

Again, even though these instances have the largest variability in the distribution of the data parameters, the results are as good as in the case of the CPARP. All the instances of this group were optimally solved at the root node, what clearly illustrates the efficiency of the algorithm.

Finally, in Table 6.19 the results for the *GRID* instances are given. The results are as good as in general for all other instances. There are 31 out of 36 optimally solved at the root node of the exploration tree. However, as can be observed in Table 6.19, Instance G32 took almost 7 hours to be solved. Besides this drawback, for all instances but G28 the relaxation gap is below 5%, which as mentioned above, is a good indicator for the algorithm.

In general terms, the above results indicate that in most cases our algorithm is able to optimally solve the instances in small times. However, it is also true that our results are not comparable with those obtained with the best specific algorithms for the RPP [11, 18, 22].

name	z^*	z_r	z_h	g_r	g_h	n	t_{z_r}	t_{z^*}
G0	12	12.00	12	0.0	0.0	0	0.015	0.031
G1	14	14.00	14	0.0	0.0	0	0.031	0.031
G2	12	12.00	12	0.0	0.0	0	0.016	0.016
G3	16	16.00	16	0.0	0.0	0	0.047	0.047
G4	16	16.00	16	0.0	0.0	0	0.265	0.281
G5	14	14.00	14	0.0	0.0	0	0.031	0.031
G6	20	20.00	20	0.0	0.0	0	0.063	0.094
G7	16	16.00	16	0.0	0.0	0	0.031	0.031
G8	18	18.00	18	0.0	0.0	0	0.047	0.047
G9	24	24.00	24	0.0	0.0	0	0.219	0.219
G10	30	29.33	30	2.3	0.0	0	0.406	0.469
G11	30	30.00	30	0.0	0.0	0	0.312	0.375
G12	42	42.00	42	0.0	0.0	0	0.297	0.297
G13	40	40.00	40	0.0	0.0	0	0.218	0.218
G14	42	40.67	42	3.3	0.0	2	0.344	0.688
G15	48	48.00	48	0.0	0.0	0	0.234	0.234
G16	46	46.00	46	0.0	0.0	0	0.375	0.391
G17	48	48.00	48	0.0	0.0	0	0.297	0.297
G18	48	48.00	48	0.0	0.0	0	0.703	0.718
G19	52	52.00	52	0.0	0.0	0	2.406	2.563
G20	54	54.00	54	0.0	0.0	0	1.719	1.938
G21	70	69.33	70	1.0	0.0	0	1.594	1.781
G22	68	68.00	68	0.0	0.0	0	1.266	1.281
G23	74	74.00	74	0.0	0.0	0	1.906	1.906
G24	90	90.00	90	0.0	0.0	0	2.094	2.094
G25	84	84.00	84	0.0	0.0	0	1.516	1.532
G26	84	84.00	84	0.0	0.0	0	1.547	1.563
G27	78	78.00	80	0.0	2.6	2	4.297	10.891
G28	86	86.00	92	0.0	7.0	53	13.594	253.797
G29	82	81.33	84	0.8	2.4	50	8.984	166.422
G30	110	109.50	110	0.5	0.0	0	20.703	21.265
G31	114	114.00	114	0.0	0.0	0	11.563	11.610
G32	120	118.50	120	1.3	0.0	4388	5.906	27558.422
G33	148	148.00	148	0.0	0.0	0	4.750	4.782
G34	144	143.33	144	0.5	0.0	0	10.750	11.813
G35	142	142.00	142	0.0	0.0	0	12.406	13.140

Table 6.19: *Solving the RPP on GRID instances with the CPARP algorithm.*

Chapter 7

The Windy CPARP

In this chapter we study the Windy Clustered Prize-collecting Arc Routing Problem (WCPARP), which is the windy version of the CPARP.

Like all other arc routing problems, the difference between the version on an undirected graph and the version on a windy graph is that in the former, the cost of traversing an edge does not depend on the direction of the traversal, whereas in the latter, the cost of traversing an edge may be different in each of its two opposite directions. Similarly to other windy arc routing problems, in the WCPARP it is possible that an optimal solution traverses an edge in the same direction several times. Therefore, a natural formulation would use general integer variables allowing this possibility. However, given the good results obtained with the binary formulation for the CPARP, we will also formulate the WCPARP using only binary variables.

In this chapter we first give the definition of the WCPARP. An example is given next. Then, we present a formulation with general integer variables corresponding to the times that each edge is traversed by solution tours in each of its two directions. This formulation is presented here since it seems to be the natural formulation for the problem. However, after that, another formulation for the WCPARP, using only binary variables, is proposed. For defining variables as binary, an upper bound on the number of times that any edge can be used is established. Thus, we can propose a formulation of the problem with binary variables by introducing as many copies of each edge as needed. As with the CPARP, for this formulation we define an equivalent problem on a complete graph, and study some properties from this new point of view. The chapter finishes with the results of a preliminary computational experience using an algorithm that implements the latter formulation, leaving the former for future research.

7.1 Definition of the WCPARP

Before proceeding, recall that $G = (V, E)$ denotes a given graph with a distinguished vertex $d \in V$ that represents the depot, and a subset $D \subset E$, $D \neq \emptyset$, representing the set of demand edges. Now, there are two costs denoted c_{uv} and c_{vu}, representing the cost of traversing edge $e = uv \in E$ from u to v and from v to u, respectively. As before, associated with each demand edge $e \in D$ there is also a value $b_e > 0$ that represents the profit for servicing edge e. Thus, we assume that the profit of each demand edge does not depend on the direction in which it is serviced, and also like before, edges without demand, $e \in E \setminus D$, are assumed to have profit zero. Again, $G_D \equiv (V(D) \cup \{d\}, D)$ denotes the subgraph induced by the edge set D and the depot, and C_k, $k \in \{0, \ldots, p\}$, are the connected components of the graph G_D.

In the WCPARP, the cost of a tour $\mathcal{T}$ accounts for all the edges that are traversed, taking into account the number of times that each edge is used in each possible direction. That is,

$$c(\mathcal{T}) = \sum_{uv \in T} (t_{uv} c_{uv} + t_{vu} c_{vu})$$

where t_{uv} and t_{vu} denote the number of times that $\mathcal{T}$ traverses edge $e = uv$ from u to v and from v to u, respectively.

Definition 7.1 The Windy Clustered Prize-collecting Arc Routing Problem (WCPARP)

Feasible solutions for the WCPARP are tours going through d, such that for each cluster C_k, $k \in \{0, \ldots, p\}$, either all its edges are serviced or none of its edges is serviced.

The Windy Clustered Prize-collecting Arc Routing Problem is to find a set of clusters $\mathcal{K}^* \subseteq \{0, \ldots, p\}$, and a tour $\mathcal{T}^*$, passing through d, that services all the edges in $\cup_{k \in \mathcal{K}^*} C_k$, but none of the edges in $D \setminus \cup_{k \in \mathcal{K}^*} C_k$, which maximizes the value of

$$\sum_{k \in \mathcal{K}} F_k - \sum_{uv \in \mathcal{T}} (t_{uv} c_{uv} + t_{vu} c_{vu})$$

over all feasible tours $\mathcal{T}$, where $\mathcal{K}$ is the set of clusters serviced in the tour $\mathcal{T}$, t_{uv} is the number of times that edge $e = uv$ is traversed from u to v in $\mathcal{T}$, and $F_k = b(C_k)$.

We represent a WCPARP by $WCPARP(G, D, d, b, c)$.

Remark that the WCPARP is $\mathcal{NP}$-hard, since there exists an easy polynomial reduction from the CPARP.

Example

In Figure 7.1(a), an instance of WCPARP is depicted. The value on edge uv next to vertex u corresponds to c_{uv}, whereas the value next to vertex v corresponds to c_{vu}. As usual, demand edges appear in bold. As can be seen, there are three clusters with sets of vertices $V_0 = \{d\}$, $V_1 = \{1, 2, 6, 7\}$, and $V_2 = \{3, 4, 5, 8, 9, 10\}$. Also, $C_1 = \{\{1, 2\}, \{1, 6\}, \{2, 7\}\}$ and, $C_2 = \{\{3, 4\}, \{4, 5\}, \{5, 10\}, \{8, 9\}, \{9, 10\}\}$. Edges $\{3, 8\}$ and $\{4, 9\}$ have its two end-nodes in V_2, but they do not have demand. All demand edges have profit $b_e = 10$.

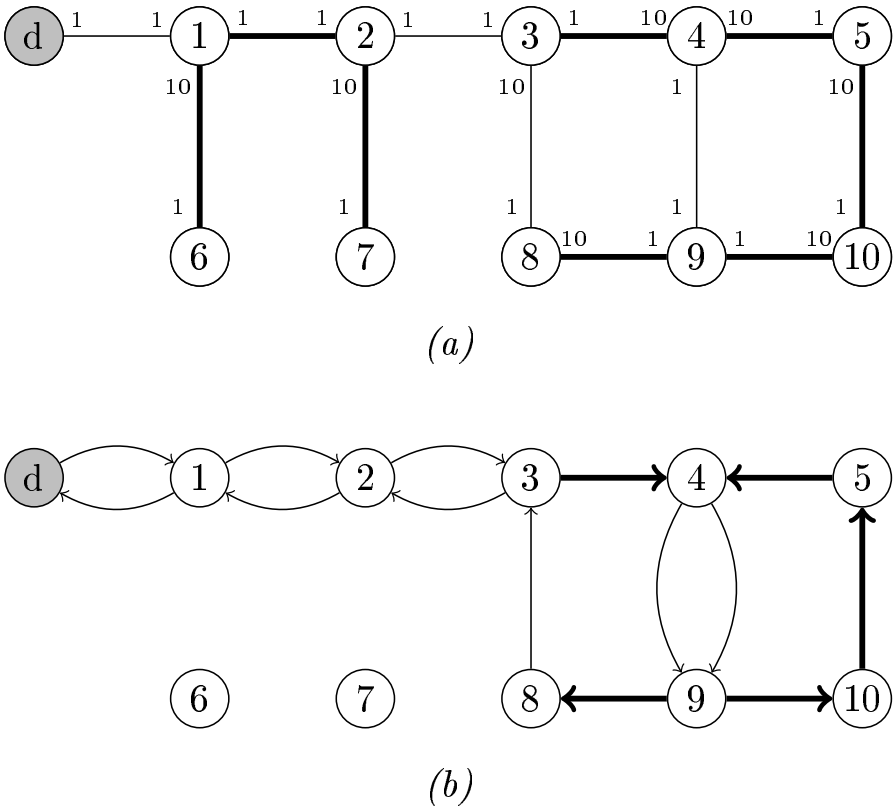

Figure 7.1: *(a) Example instance. Both costs are shown for all edges (the nearest to a node is its outgoing cost). Demand edges are in bold, with $b_e = 10$ for all of them; (b) Optimal directed solution giving service to C_2, with value $z^*_{WCPARP} = 36$. Only serviced edges in bold.*

In Figure 7.1(b) the unique optimal solution is shown. Now solutions are directed graphs. The only cluster serviced by the solution is C_2. Only serviced edges are in bold. Observe that edges {1,6} and {2,7} are two demand edges not even traversed. Another demand edge, {1,2}, is traversed but not serviced. Also, attention must be paid to the fact that the solution is using twice edge {4,9} in its forward direction. The value of the solution is $z^*_{WCPARP} = 50 - 14 = 36$.

7.2 Formulation of the WCPARP with integer variables

Throughout this section we will work on a simplified graph $G' = (V(D), E')$ obtained from $G = (V, E)$ similarly as for the Windy Rural Postman Problem in [30, 49]. The set E' contains the edges in D plus some other representing minimum cost paths in the original graph. In particular, E' is obtained by first adding to D an edge uv for each pair of vertices $u, v \in V(D)$ with cost values, c_{uv} and c_{vu}, equal to the costs of the minimum cost paths in G from u to v, and from v to u, respectively.

Then, for each edge $e = uv \in E'$ we define two variables x_{uv} and x_{vu} representing the number of times edge e is traversed from u to v and from v to u, respectively. In this manner, for a vertex $u \in V(D)$ we can denote $x(\delta^+(u))$ and $x(\delta^-(u))$ to the sums of its outgoing and incoming traversals, respectively. This is, $x(\delta^+(u)) = \sum x_{uv}$, and $x(\delta^-(u)) = \sum x_{vu}$. We also define $p + 1$ additional binary variables z_k, $k \in \{0, \ldots, p\}$, that take value one if component C_k is serviced in the solution tour, and zero otherwise.

The problem can be formulated as follows:

$$(W1) \qquad \max \quad \sum_{k=0}^{p} F_k z_k - \sum_{e \in E'} (c_{uv} x_{uv} + c_{vu} x_{vu}) \tag{7.1}$$

$$x_{uv} + x_{vu} \geq z_k, \qquad k = \{0, \cdots, p\}, uv \in C_k \tag{7.2}$$

$$x(\delta^+(u)) = x(\delta^-(u)), \qquad u \in V(D) \tag{7.3}$$

$$x(\delta(S)) \geq 2z_k, \qquad S \subset \cup_{k \in \pi} V_k, \pi \in \mathcal{P}(\Omega_p) \tag{7.4}$$

$$x_{uv}, x_{vu} \geq 0 \text{ and integer}, uv \in E' \tag{7.5}$$

$$z_k \in \{0, 1\}, \ k \in \{0, \cdots, p\} \tag{7.6}$$

where $\Omega_p = \{1, \ldots, p\}$.

Inequalities (7.2) guarantee that the route traverses all the demand edges of the components that it serves; Equations (7.3) force the route to be symmetric, by guaranteing that for each vertex $u \in V(D)$ the number of edges of the route incoming vertex u is the same as the number of edges of the route outgoing vertex u. Inequalities (7.4) ensure that the route connects the edges it serves and the depot. Observe that, because of constraints (7.3), each constraint (7.4) is equivalent to the pair of inequalities $x(\delta^+(S)) \geq z_k$, $x(\delta^-(S)) \geq z_k$. Finally, the domain for all variables is established in the expressions (7.5) and (7.6).

The formulation of WCPARP with integer variables will be studied in future research.

7.3 Formulation of the WCPARP with binary variables

Looking forward to making use of the cocircuit inequalities shown in Subsection 3.4.2, we have adressed the study of the formulation with binary variables.

Analogously to the case of the CPARP, we are going to work on the complete graph K_n defined on the set of nodes $V(D) \cup \{d\}$. We propose a formulation on the $WCPARP(K_n, D, d, b, c)$ that is equivalent to the problem on the original graph, $WCPARP(G, D, d, b, c)$, in the sense that optimal solution values coincide. In the first subsection the transformation is detailed. Next, several dominance relations are proved, and the sets of variables are defined based on these properties. An ILP formulation for the WCPARP with binary variables is then presented, whose LP relaxation has been tested in the computational experiments of the WCPARP, obtaining hereby an upper bound for the integer optimal value of each problem. These results are seen in the next section.

7.3.1 Graph transformation: WCPARP on K_n

Let us consider the complete graph K_n, with vertex set $V(D) \cup \{d\}$, and therefore $n = |V(D) \cup \{d\}|$. To simplify the study, abusing slightly notation, from now on we will denote $V := V(D) \cup \{d\}$.

For each pair $u, v \in V$, let P^G_{uv} and P^G_{vu} respectively denote the minimum cost paths in G from u to v and from v to u. Then, $c(P^G_{uv})$ and $c(P^G_{vu})$ denote their values relative to the cost function. The definition of the profit and cost functions on K_n follows a rationale quite similar to that used in the transformation of the CPARP of Subsection 3.2.1, except that now we define the transformation for both costs.

Dealing with edges that do not fulfill the triangular inequality with respect to the cost function, now the situation is slightly more complicated than in the CPARP seen in Subsection 3.2.1, since an edge $uv \in E$ might fulfil the triangular inequality in just one direction. In general, the values $\Delta_{uv} = c_{uv} - c(P^G_{uv})$ and $\Delta_{vu} = c_{vu} - c(P^G_{vu})$ need not coincide in the original graph. When $e = uv \in D$ has $\Delta_{uv} > 0$, then, in any optimal solution edge e will be traversed in the direction from u to v (at most) only when it is serviced, with a net profit $b_e - c_{uv} = (b_e - \Delta_{uv}) - (c_{uv} - \Delta_{uv})$. All other times that an optimal tour goes from u to v in the original graph, the path P^G_{uv} will be used with cost $c(P^G_{uv}) = c_{uv} - \Delta_{uv}$. Taking this observation into account, the cost functions for an edge $e = uv \in E(K_n)$ in the problem stated on K_n are defined as

- $c_{uv} := c(P^G_{uv})$, when $e = uv$ is traversed from u to v, and

- $c_{vu} := c(P^G_{vu})$, when $e = uv$ is traversed from v to u.

Moreover, when $e \in D$, the profit is

- $b_{uv} := b_{uv} - \Delta_{uv}$, when $e = uv$ is serviced in the direction from u to v, or
- $b_{vu} := b_{vu} - \Delta_{vu}$, when $e = uv$ is serviced in the direction from v to u.

Like in Chapter 3, let H denote the set of non demand edges in K_n, and again, we partition the set H of non demand edges into $H_1 \cup H_2$, where H_1 denotes the set of non-demand edges with the two end-nodes in the same cluster, and H_2 denotes the set of edges with the two end-nodes in different clusters.

From the above definitions it is easy to see that any feasible solution $\mathcal{T}_{K_n}$ to the transformed problem $WCPARP(K_n, D, d, b, c)$ defines a feasible solution $\mathcal{T}_G$ to the original problem $WCPARP(G, D, d, b, c)$, which can be obtained by substituting any edge $e \in H$ of $\mathcal{T}_{K_n}$, by the edges of the path P^G_e. The values of $\mathcal{T}_{K_n}$ and $\mathcal{T}_G$ coincide. Moreover, any feasible solution, $\mathcal{T}_G$, to the original $WCPARP(G, D, d, b, c)$ is also a feasible solution to the transformed $WCPARP(K_n, D, d, b, c)$, although their values do not necessarily coincide. Anyway, for the case of optimal solutions, if $\mathcal{T}^*_G$ is an optimal solution to the $WCPARP(G, D, d, b, c)$, any chain of traversed-but-not-serviced edges defines a shortest path between its end-nodes, P^G_e, so that the value of $\mathcal{T}^*_G$ coincides with the optimal value of the solution on the complete graph, $\mathcal{T}^*_{K_n}$.

As a consequence, any optimal solution to the $WCPARP(K_n, D, d, b, c)$ defines an optimal solution to $WCPARP(G, D, d, b, c)$.

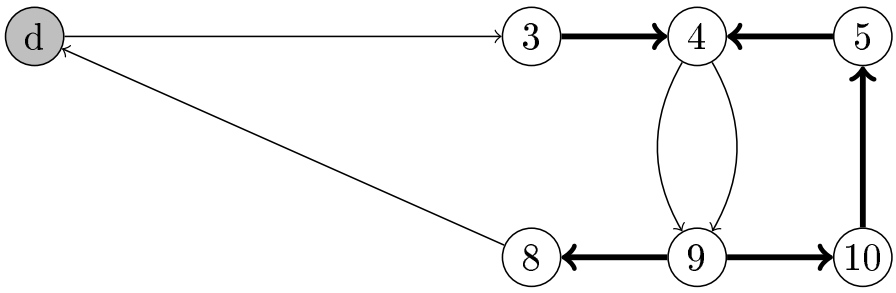

Figure 7.2: *Optimal solution for the instance of Figure 7.1(a) when the problem is stated on the complete graph.*

An optimal solution to the example of Figure 7.1(a) after the transformation of the problem to the problem stated on the complete graph is depicted in Figure 7.2. As it can be seen, in this example any optimal solution traverses two times edge $\{4, 9\}$ in the same direction. Indeed, this represents a difficulty for formulating the problem using only binary variables. We next study some

properties of $WCPARP(K_n, D, d, b, c)$ in terms of dominance relations that are useful towards this goal.

7.3.2 Properties of the WCPARP

Some dominance relation are listed below. They will we be used for the formulation of the WCPARP with binary variables.

We say that an ordered pair of edges, $e - f$ such that $e, f \in \delta(u)$, $u \in V$ are *consecutive* in a feasible solution T_{K_n} if both edges are used in T_{K_n}, e in the direction incoming node u, and f in the direction outgoing node u. Also, throughout, it must be clearly distinguished between servicing an edge, and traversing it. Thus, we are going to use the term *traverse* for making reference to use an edge without giving service to it.

Dominance 7.1 *There exists an optimal solution $\mathcal{T}^*$ to $WCPARP(K_n, D, d, b, c)$ that does not traverse any pair of consecutive edges.*

Proof *Suppose there exists a pair of consecutive edges $e - f$, $e = uv$, and $f = uw$, $u, v, w \in V$, that are traversed in $\mathcal{T}^*$. Then, the traversal of the consecutive pair $e - f$ can be substituted by one traversal of edge vw in the direction outgoing node v, without deteriorating the objective function value.* □

From Dominance 7.1, an optimal solution to $WCPARP(K_n, d, D, b, c)$ exists, in which every time that edge $e = uv$ is used in the direction from u to v either e is serviced, or e is traversed as a means of connecting two edges that are serviced, one in the direction incoming node u and the other one in the direction outgoing node v. Thus, similarly to Dominance 3.6 of Chapter 3 for the CPARP, we also have the following dominance relation.

Dominance 7.2 *There exists an optimal solution to $WCPARP(K_n, D, d, b, c)$ that does not traverse any edge of any non serviced component.*

Note that as a consequence of dominance relations 7.1 and 7.2, there exists an optimal solution to $WCPARP(K_n, D, d, b, c)$ in which no edge in the cut-set of a non serviced component is traversed, as already was said in Dominance 3.6 for the CPARP.

Next dominance relation is later used for upper bounding the number of binary variables associated with each edge. Indeed, we refer to the number of uses of each edge by optimal solution tours.

Dominance 7.3 *There exists an optimal solution to* $WCPARP(K_n, d, D, b, c)$ *in which for any edge* $e = uv$ *the maximum number of times that* e *is used in each of its two possible directions is given by* $U_e = \min\{|\delta_D(u)|, |\delta_D(v)|\}$.

Proof *Let* $\mathcal{T}^*$ *be an optimal solution to* $WCPARP(K_n, D, d, b, c)$ *in which* $e = uv$ *is used more than* U_e *times in the direction from* u *to* v. *Let also* $t_{uv} > U_e$ *denote the number of times that* e *is traversed in* $\mathcal{T}^*$ *in the direction from* u *to* v. *Without loss of generality also assume that* $U_e = |\delta_D(u)|$. *Since* $t_{uv} > |\delta_D(u)|$, *there must exist* $f = uw, f \notin D$, *which is traversed in* $\mathcal{T}^*$ *in the direction from node* w *to node* u. *Thus, one traversal of the consecutive pair of edges* $f - e$ *can be substituted by one traversal of edge* wv *in the direction incoming node* v, *without deteriorating the objective function value.* □

When needed, we will use $U_{uv} \equiv U_e$.

As opposed to the CPARP, in the WCPARP there might exist non demand edges in H_1 with some D-even end-node that must be traversed in any optimal solution. This is illustrated by edge $\{4, 9\}$ in the solution of Figure 7.1(b) of page 101, or also in Figure 7.2.

We now turn our attention to the edges that are used in both directions in optimal solutions to the $WCPARP(K_n, D, d, b, c)$. For this kind of edges we will find some similarities with the CPARP. In Dominance 7.4, edges that can be used in both directions in optimal solutions are identified. As can be observed, the sets of such edges are defined analogously as those used for the CPARP in Preprocessing 3.9.

For Preprocessing 7.4, recall the sets of edges defined in Subsection 3.2.2. In particular, D_o was defined as the set of edges with both end-nodes D-odd, and also M_0 as the set of edges in H_2 (so connecting different clusters) with both end-nodes D-even, such that their costs are minimum among all edges connecting the same pair of clusters through D-even vertices. Now these minimum costs must be understood as the sum of both costs for each edge.

Preprocessing 7.4 *There is an optimal solution to* $WCPARP(K_n, d, D, b, c)$ *where the only edges that are used in both directions are either* $e \in D \cap D_o$ *or* $e \in M_0$.

Proof *Let* $\mathcal{T}^*$ *be an optimal solution in which edge* $e = uv$ *is used in both directions.*

1. *Let* $e \in D$. *Suppose vertex* u *is* D*-even and suppose, without loss of generality, that edge* e *is serviced in the direction from* u *to* v. *Edge* e *is traversed therefore in the direction from* v *to* u. *Hence, there exists* $f = uw$ *which is traversed in* $\mathcal{T}^*$ *in the direction outgoing node* u. *Therefore, one traversal of the consecutive pair of edges* $e - f$ *can be substituted by one*

traversal of edge vw in the direction outgoing node v, without deteriorating the objective function value.

2. *Let $e \in H_2$. Suppose vertex u is D-odd. Since edge e is traversed in the direction from v to u there exists $f = uw$ which is traversed in $\mathcal{T}^*$ in the direction outgoing node u. Therefore, one traversal of the consecutive pair of edges $e - f$ can be substituted by one traversal of edge $\{v, w\}$ in the direction outgoing node v, without deteriorating the objective function value. As a consequence, for $e \in H_2$ only those with both vertices D-even are candidates to be used twice in optimal solutions. Consequently, they will belong to the set M_0.*

Observe finally that we can assume that no edge $e \in H_1$ is traversed in both directions in $\mathcal{T}^*$, since the removal of one traversal in each direction will produce a feasible solution with a value strictly better than that of $\mathcal{T}^*$.

□

Next, we further analyze the edges that can be used in both directions and did not satisfy the triangular inequality in the original graph G. Recall from Subsection 7.3.1 that in the transformed complete graph not only the cost function is asymmetric, but also the profit function. Therefore, for demand edges, $e = uv \in D$, that do not satisfy the triangular inequality in the original graph G, the contribution to the objective function is not the same when service takes place in the direction from u to v, with a total contribution of $(b_{uv} - c_{uv}) - c_{vu} = b_{uv} - (c_{uv} + c_{vu})$, than when service takes place from v to u, with a total contribution of $(b_{vu} - c_{vu}) - c_{uv} = b_{vu} - (c_{uv} + c_{vu})$. That is, we can assume that in any optimal solution that uses edge $e = uv$ in both directions, service of edge e takes place in the direction corresponding to $\max\{b_{uv}, b_{vu}\}$ (with ties broken arbitrarily). Also recalling from Subsection 7.3.1 the definition of $\Delta_{uv} = c_{uv} - c(P^G_{uv})$, the best direction is the one that gives $\min\{\Delta_{uv}, \Delta_{vu}\}$. In other words, service will be given in the direction which, in terms of the original graph G, produces a smaller saving for using the shortest path instead of the original link. For denoting such direction, associated with each edge $e = uv$, let the ordered pair (u_e, v_e) be defined as follows

$$(u_e, v_e) = \begin{cases} (u, v) & \text{if } \Delta_{uv} \geq \Delta_{vu}, \\ (v, u) & \text{otherwise,} \end{cases}$$

For unifying purposes we arbitrarily define the direction of the first traversal of a non-demand edge $e = uv \in H$ as $(u_e, v_e) = (u, v)$.

The definitions stated above allow us to establish next dominance relation.

Dominance 7.5 *There exists an optimal solution to $WCPARP(K_n, D, d, b, c)$, in which any edge $e = uv$ that is used in both directions is serviced in the direction from u_e to v_e.*

It is clear that when an edge is used in both directions, it will be used exactly once in, at least, one direction. Otherwise the removal of one traversal of each direction would produce a feasible solution with an improved value. If, moreover, we deal with a non-demand edge, then it will be traversed exactly once in both directions by the same reason.

From the reasons explained above, it results that a demand edge might not be serviced the first time it is traversed. This might be the most significant difference between the WCPARP and its predecessor, the CPARP.

7.3.3 Definition of variables

Even though for the case of the WCPARP they do not appear in the objective function, $p+1$ variables z_k are defined as in the case of the CPARP, indicating whether or not the optimal solution gives service to cluster k, $k \in \{0, \ldots, p\}$.

Also, for each edge $e = uv \in E(K_n)$, we define two sets of x variables, x^t_{uv}, and x^t_{vu}, $t = 1, \ldots, U_{uv}$, where, as defined in Dominance 7.3, $U_e = U_{uv} = \min\{|\delta_D(u)|, |\delta_D(v)|\}$.

They are defined as follows:

$$x^1_{uv} = \begin{cases} 1 & \text{if edge } uv \in D \text{ is serviced the first time it is used in the direction} \\ & \text{from } u \text{ to } v, \\ 1 & \text{if the first time that edge } uv \in H \text{ is traversed, it is so from } u \text{ to } v, \\ 0 & \text{otherwise.} \end{cases}$$

Taking into account that non-demand edges have profit zero, the coefficient in the objective function of these variables is $b_{uv} - c_{uv}$.

For $1 < t \leq U_e$,

$$x^t_{uv} = \begin{cases} 1 & \text{if edge } uv \text{ is traversed a } t\text{-th time from } u \text{ to } v, \\ 0 & \text{otherwise.} \end{cases}$$

The coefficient in the objective function of these variables is $-c_{uv}$.

For simplicity of notation, in what follows, for any subset $S \subset V$, $S \neq \emptyset$ we will use

$$x(\delta^+(S)) = \sum_{uv \in \delta(S)} \sum_{t=1}^{U_{uv}} x_{uv}^t \tag{7.7}$$
$$x(\delta^-(S)) = \sum_{uv \in \delta(S)} \sum_{t=1}^{U_{uv}} x_{vu}^t$$
$$x(\delta(S)) = x(\delta^+(S)) + x(\delta^-(S))$$

As we have previously observed, some demand edges are not serviced the first time they are used in a given direction. These are the edges that are used in both directions and that are serviced in the opposite direction. Recall that the edges that can be used in both directions have been determined in Dominance 7.4 to belong to the sets $D \cap D_o$ and M_0 for demand and non demand edges, respectively.

Recall also that Dominance 7.5 indicates that for such edges the service direction (fictitious in the case of non-demand edges) is (u_e, v_e). Thus, we define one additional variable $y_{v_e u_e}$ for each edge

$$y_{v_e u_e} = \begin{cases} 1 & \text{if edge } e \in (D \cap D_o) \cup M_0 \text{ is used in both directions and} \\ & \text{the first use has been from } v_e \text{ to } u_e, \\ 0 & \text{otherwise.} \end{cases}$$

For the y variables an analogous notation for the expressions (7.7) is used, even though all of them involve just one y_{uv} for each edge. This is,

$$y(\delta^+(S)) = \sum_{uv \in \delta(S)} y_{uv}$$
$$y(\delta^-(S)) = \sum_{uv \in \delta(S)} y_{vu}$$
$$y(\delta(S)) = y(\delta^+(S)) + y(\delta^-(S))$$

7.3.4 Objective Function

As opposed to the CPARP, in the WCPARP defined on a complete graph we do not know beforehand the gross benefit of servicing a given cluster. This leads not to include in the objective function the variables associated with the service

of clusters. For this reason, the objective function is

$$\max\ Z = \sum_{uv\in E}(b_{uv}-c_{uv})x^1_{uv} - \sum_{uv\in E} c_{uv}y_{uv} - \sum_{uv\in E}\sum_{t=2}^{U_e} c_{uv}x^t_{uv} \qquad (7.8)$$

where after the transformation we denote E to the set of edges of K_n, that is $E := E(K_n)$.

7.3.5 ILP01 Formulation

The collection of constraints of the polyhedral model for the WCPARP is split into five classes which are enumerated next with a brief semantic annotation for each one of them.

- *Mandatory:* Characterizing the problem as clustered. Inequalities (7.9) impose the clustering restriction on the demand edges. Inequalities (7.10) are implied by Dominances 7.1 and 7.2. They forbid using a vertex of a non serviced cluster.

$$x^1_{uv} + x^1_{vu} = z_k, \qquad uv \in C_k,\ \ k \in \{0,\dots,p\} \quad (7.9)$$
$$x^1(\delta_D(u)) + y(\delta_D(u)) \geq x^1(\delta(u)\setminus\delta_D(u)) + y(\delta(u)\setminus\delta_D(u)),\ u \in V \quad (7.10)$$

- *Domain:* Establishing dominance relations among variables. Depending on the type of variable, Inequalities (7.11), forbid the *first* use of an edge in both directions. Inequalities (7.12), for variables y_{v_e,u_e} seen in Dominance 7.5, mean that a first use is required for going back through a given edge. Inequalities (7.13) are related to the t-th traversal, with $t > 1$, and they impose that edges must have been traversed $t-1$ times before their t-th traversal.

$$x^1_{uv} + x^1_{vu} \leq 1, \qquad uv \in E \qquad (7.11)$$
$$x^1_{uv} - y_{vu} \geq 0, \qquad uv \in E_y \qquad (7.12)$$
$$x^t_{uv} \leq x^{t-1}_{uv}, \qquad t = \{2,\dots,U_{uv}\}, uv \in E \qquad (7.13)$$

  where $E_y = (D \cap D_o) \cup M_0$.

- *Symmetry:* Ensuring the parity of the vertices. Equations (7.14) are commonly used in most of windy problems, they impose that the number of times the solution tour gets into a vertex is that one of times it gets out of it.

$$x(\delta^+(u)) + y(\delta^+(u)) = x(\delta^-(u)) + y(\delta^-(u)), \qquad u \in V \qquad (7.14)$$

- *Connectivity:* Inequalities (3.8) for the CPARP are repeated in (7.15).

$$x(\delta(\bigcup_{k\in\pi} V_k)) + y(\delta(\bigcup_{k\in\pi} V_k)) \geq 2z_k, \qquad k \in \pi,\ \pi \in \mathcal{P}(\Omega_p) \qquad (7.15)$$

where $\Omega_p = \{1, \ldots, p\}$. Working on the WCPARP we realized that they can be strengthened by imposing that for each $k \in \pi, \pi \in \mathcal{P}(\Omega_p)$

$$x(\delta(\underset{k\in\pi}{\cup} V_k)) \geq z_k \tag{7.16}$$
$$x(\delta(\underset{k\in\pi}{\cup} V_k)) + y(\delta(\underset{k\in\pi}{\cup} V_k)) \geq 2z_k$$

In fact, this would mean an improvement to the formulation of the CPARP of Subsection 3.4.4. Thus, for the case of the WCPARP we introduced the connectivity constraints in the form of Inequalities (7.16).

- *Cocircuit:* Reinforcing the symmetry constraints for sets of vertices.

$$x(\delta(S) \setminus F) + y(F \setminus L) \geq x(F) + y(L) - (|F| + |L|) + 1 \tag{7.17}$$
$$S \subset V(D) \cup \{d\},\ F \subseteq \delta(S),\ L \subseteq F \cap E_y, (|F| + |L|) \text{ odd},$$

where $E_y = (D \cap D_o) \cup M_0$ as for the case of the CPARP, in the Expression 3.3.

Summarizing all the expressions seen above, a binary formulation for the WCPARP is next proposed.

(W2)

$$\max\ Z = \sum_{uv\in E} (b_{uv} - c_{uv})x^1_{uv} - \sum_{uv\in E} c_{uv}y_{uv} - \sum_{uv\in E} \sum_{t=2}^{U_e} c_{uv}x^t_{uv} \tag{7.18}$$

$$x^1_{uv} + x^1_{vu} = z_k, \qquad uv \in C_k,\ \ k \in \{0, \ldots, p\} \tag{7.19}$$
$$x^1(\delta_D(u)) + y(\delta_D(u)) \geq x^1(\delta(u) \setminus \delta_D(u)) + y(\delta(u) \setminus \delta_D(u)),\ u \in V \tag{7.20}$$

$$x^1_{uv} + x^1_{vu} \leq 1, \qquad uv \in E \tag{7.21}$$
$$x^1_{uv} - y_{vu} \geq 0, \qquad uv \in E \tag{7.22}$$
$$x^t_{uv} \leq x^{t-1}_{uv}, \qquad t = \{2, \ldots, U_{uv}\}, uv \in E \tag{7.23}$$

$$x(\delta^+(u)) + y(\delta^+(u)) = x(\delta^-(u)) + y(\delta^-(u)), \qquad u \in V \tag{7.24}$$

$$x(\delta(\underset{k\in\pi}{\cup} V_k)) \geq z_k, \qquad k \in \pi, \pi \in \mathcal{P}(\Omega_p) \tag{7.25}$$
$$x(\delta(\underset{k\in\pi}{\cup} V_k)) + y(\delta(\underset{k\in\pi}{\cup} V_k)) \geq 2z_k, \qquad k \in \pi, \pi \in \mathcal{P}(\Omega_p)$$

$$x(\delta(S) \setminus F) + y(F \setminus L) \geq x(F) + y(L) - (|F| + |L|) + 1, \tag{7.26}$$
$$S \subset V(D) \cup \{d\},\ F \subseteq \delta(S),\ L \subseteq F \cap E_y, (|F| + |L|) \text{ odd}$$

$$z_k \in \{0, 1\}, \qquad k \in \{0, \ldots, p\} \tag{7.27}$$
$$x^t_e \in \{0, 1\}, \qquad t \in \{1, \ldots, U_e\}, e \in E \tag{7.28}$$
$$y_e > 0, \qquad e \in E_y \tag{7.29}$$

where $\Omega_p = \{1, \ldots, p\}$.

Note that Proposition 3.11 of page 43 for the CPARP is also valid for the WCPARP. Thus, Inequalities (7.29) are due to the result of that proposition.

7.3.6 Algorithm

In this section we propose an iterative algorithm to obtain upper bounds for the optimal solutions to the formulation W2 seen in the previous section. We focus on the linear problem relaxation of that formulation.

For solving the LP relaxation to the formulation W2, we start by omitting the constraints related to the integrity of the variables, which are Inequalities (7.27) and (7.28). Instead, we impose an upper bound of 1 on the corresponding continuous variables.

Like in the CPARP there are several families of inequalities which have an exponential size on several parameters. Therefore, exactly as for the case of the CPARP, we face the problem using an iterative methodology. Next, we first explain which are the constraints introduced in the initial model, and after, the differences in the separation procedures with respect to the CPARP.

Initial Formulation

For the computational experiments, we have built an initial model in which we have introduced a small part of the constraints of Subsection 7.3.5. These consist of the three first sets (*Mandatory*, *Domain* and *Symmetry* constraints). With respect to the connectivity inequalities, as in the CPARP, all Inequalities (7.25) such that $|\pi| = 1$ have been included in the initial problem. In this way, we are keeping in $\mathcal{O}(n)$ the number of constraints introduced at the first stage. Finally, with respect to the cocircuit inequalities, we proceeded exactly as for the case of the CPARP. This is, we have omitted them in the initial formulation. Given that we have already the symmetry constraints for the vertices, we reinforced the parity restrictions by including a new inequality for each D-odd vertex, meaning that some non-demand traversal must be added to the cut of those vertices if its corresponding cluster is serviced. Therefore, the semantic interpretation of the constraint is exactly the same than in the CPARP. However, we have to express this restriction using the new notation for the sets.

$$x(\delta(u)) + y(\delta(u)) - x(\delta_D(u)) \geq z_k,\ u \in V_k, k \in \{0, \ldots, p\}, |\delta_D(u)| \text{ odd} \quad (7.30)$$

Separation of Inequalities

To separate connectivity constraints for the WCPARP we have adapted the same procedures of the CPARP, taking into account the variables associated with the t-th traversals of the edges. The values corresponding to these variables have simply been added when assigning the capacities of the maxflow problem instance.

For the cocircuit inequalities, in the WCPARP some detail must be considered that makes the procedure slightly complicated. With respect to the sets F and L (and especially to their cardinality, $|F|$ and $|L|$), each traversal t of each edge has been counted as an element of these sets. This is, given an edge and t traversals of it, $x^t_{uv} + x^t_{vu}$ must be considered as an element of the sets F or L when counting their cardinalities.

7.4 Computational Results

Two sets of preliminary results are presented in this section. First, we tested the algorithm on a workbench of instances obtained from the CPARP ones used in Chapter 6 by assigning random costs for both directions to the edges. For these instances we only can know about their optimality for those cases in which the solution to the LP relaxation is integer. Given that no heuristic is available, we can not know about the gaps that would illustrate the quality of the non-optimal solutions.

After that, given that an obvious polynomial reduction from the CPARP to the WCPARP consists of duplicating the costs, we have proceed by solving the CPARP instances using the WCPARP algorithm. Even though clearly this is not the purpose of the WCPARP algorithm, solving the CPARP instances with it has been useful to acquire knowledge of the quality of the LP non-optimal solutions.

7.4.1 Solving the WCPARP

Exactly as for the CPARP, for the WCPARP we must construct somehow a workbench for testing the implemented algorithm. Denoting c_{uv} and c_{vu} the costs of the new WCPARP instances, and c_e the costs of the old CPARP instances, we followed the guideline

$$c_{uv}, c_{vu} \sim U[c_e/2, 3c_e/2].$$

Then, the nearest integers to these values have been used. Thus, we work with integer values for costs and profits. The obtained results are given in Ta-

bles 7.1-7.5. The column headings coincide with those explained in Table 6.7 of Subsection 6.2.1 (with Z_w instead of Z_r). In Table 7.1, the values for the two ALBAIDA instances are presented.

name	Z_w	t_{Z_w}	Z_0	iter	con($\leq$)	coc ($\leq$)	
ALBAIDAA	5580.50	28.359	5949.50	10	4 (28)	6 (14)	
ALBAIDAB	3971.00	25.016	4253.50	10	7 (47)	3 (9)	

Table 7.1: *LP solutions of the windy ALBAIDA instances.*

Even though there are not optimal solutions obtained with these instances, the solution value to the first iteration, Z_0, is quite close to its final value Z_w. However, as can already be seen with these instances, the required times significantly grow, with respect to the CPARP, when introducing new variables.

In Table 7.2 the values for the *CHRISTOFIDES* instances are listed.

name	Z_w	t_{Z_w}	Z_0	iter	con($\leq$)	coc ($\leq$)	
P01	3.00	0.016	3.00	0	0 (0)	0 (0)	✓
P02	23.00	0.078	42.00	3	2 (5)	1 (2)	
P03	73.50	0.593	76.00	4	0 (0)	4 (8)	
P04	46.50	0.203	49.00	3	1 (2)	2 (6)	
P05	22.00	0.125	24.00	2	1 (3)	1 (2)	✓
P06	60.00	0.188	60.50	1	0 (0)	1 (3)	
P07	84.00	0.156	86.00	1	1 (2)	0 (0)	✓
P08	88.00	0.094	88.50	1	0 (0)	1 (3)	
P09	48.00	0.063	50.25	2	1 (2)	1 (1)	
P10	39.50	0.047	47.50	2	1 (2)	1 (2)	
P11	14.00	0.015	14.00	0	0 (0)	0 (0)	✓
P12	13.00	0.016	14.00	2	1 (2)	1 (2)	✓
P13	4.00	0.000	4.00	0	0 (0)	0 (0)	✓
P14	111.00	0.656	118.50	4	1 (2)	3 (7)	
P15	10.00	0.109	10.00	0	0 (0)	0 (0)	✓
P16	100.50	0.687	103.25	2	1 (2)	1 (4)	
P17	34.00	0.297	36.33	7	4 (11)	3 (3)	
P18	7.00	0.360	29.00	5	5 (15)	0 (0)	
P19	52.00	0.907	59.00	3	2 (7)	1 (1)	✓
P20	231.00	7.078	241.00	6	3 (12)	3 (8)	
P21	249.83	7.813	261.50	8	5 (16)	3 (11)	
P22	400.75	5.782	412.00	6	3 (10)	3 (13)	
P23	326.50	8.344	336.00	7	5 (16)	2 (6)	
P24	173.17	2.797	187.00	5	3 (15)	2 (6)	

Table 7.2: *LP solutions of the windy CHRISTOFIDES instances.*

There are 8 out of 21 instances optimally solved. All of them required less

than 10 seconds, whereas all but the biggest five ones required less than one second. The percent ratio between the solution value to the first iteration and the final one (this is, $100Z_0/Z_w$) is in all cases under a 133%, and in all but three cases under a 115%.

name	Z_w	t_{Z_w}	Z_0	iter	con($\leq$)	coc ($\leq$)	
D0	91.00	0.047	91.00	0	0 (0)	0 (0)	✓
D1	0.00	0.031	0.00	0	0 (0)	0 (0)	✓
D2	110.00	0.031	110.00	0	0 (0)	0 (0)	✓
D3	91.00	0.062	91.00	0	0 (0)	0 (0)	✓
D4	0.00	0.032	0.00	0	0 (0)	0 (0)	✓
D5	198.00	0.031	209.00	1	0 (0)	1 (2)	✓
D6	97.00	0.032	97.00	0	0 (0)	0 (0)	✓
D7	0.00	0.062	98.00	3	3 (8)	0 (0)	✓
D8	254.00	0.094	286.50	3	2 (5)	1 (1)	✓
D9	51.33	0.187	80.50	2	1 (2)	1 (2)	
D10	0.00	0.156	1.00	1	1 (2)	0 (0)	✓
D11	26.00	1.047	153.00	7	7 (38)	0 (0)	✓
D12	84.50	0.234	108.00	2	0 (0)	2 (3)	
D13	439.50	0.391	479.00	4	4 (12)	0 (0)	
D14	399.33	1.750	464.50	9	7 (29)	2 (9)	
D15	444.50	0.687	468.50	2	0 (0)	2 (7)	
D16	632.50	0.782	699.50	4	1 (4)	3 (4)	
D17	727.00	0.594	741.50	1	0 (0)	1 (4)	
D18	164.00	2.031	247.00	6	5 (34)	1 (1)	
D19	227.00	2.453	278.50	7	5 (26)	2 (2)	
D20	64.00	1.813	192.50	6	4 (27)	2 (2)	
D21	596.00	4.906	665.50	8	6 (18)	2 (10)	
D22	586.00	3.187	658.00	3	2 (13)	1 (10)	
D23	596.83	3.985	664.75	7	5 (16)	2 (6)	
D24	1072.50	3.625	1079.13	3	1 (2)	2 (7)	
D25	844.50	4.234	891.00	3	1 (8)	2 (9)	
D26	1134.50	5.063	1266.33	4	2 (4)	2 (17)	
D27	110.00	381.516	367.00	63	61 (758)	2 (9)	
D28	349.00	42.313	470.50	27	26 (297)	1 (4)	
D29	112.50	19.093	200.25	12	11 (112)	1 (1)	
D30	758.42	80.125	1077.50	26	23 (157)	3 (17)	
D31	971.05	32.578	1089.50	11	8 (52)	3 (20)	
D32	615.43	17.891	668.25	5	3 (11)	2 (13)	
D33	1402.22	35.422	1464.13	9	5 (27)	4 (27)	
D34	1226.00	51.375	1361.25	8	6 (32)	2 (24)	
D35	1446.83	11.313	1520.00	2	0 (0)	2 (15)	

Table 7.3: *LP solutions of the windy DEGREE instances.*

The results for the *DEGREE* instances are shown in Table 7.3. Here we find 11 optimal solutions. In fact, being optimal seems to be closely related to the size of the graph. However, as will be seen later, the algorithm finds some optimal solutions to the biggest instances. Anyway, the number of iterations is, in general, reasonably small. In all but three cases the problem is solved in

less than 13 iterations. Attention must be paid to Instance D27, which required more than five minutes to be solved. However, the same instance already needed this time when solved as an undirected CPARP. All instances but D27 run in less than 100 seconds, that are reasonable required times.

For these problems, the percent ratio between the solution value to the first iteration and the final one is under a 150% in all but three instances, and under a 133% in all but six of them. Instance D11 might also be considered as a pathological case, since the value of its first solution is five times the value of the last one.

In Table 7.4 the values obtained for the *RANDOM* instances are presented. There are 16 out of 20 instances optimally solved.

name	Z_w	t_{Z_w}	Z_0	iter	con($\leq$)	coc ($\leq$)	
R0	0.00	0.032	0.00	0	0 (0)	0 (0)	✓
R1	0.00	0.031	1761.00	1	1 (2)	0 (0)	✓
R2	0.00	0.031	2988.00	1	1 (2)	0 (0)	✓
R3	5970.00	0.032	6941.00	1	1 (2)	0 (0)	✓
R4	7545.00	0.062	10102.00	3	1 (3)	2 (2)	✓
R5	0.00	0.078	1769.00	1	1 (2)	0 (0)	✓
R6	5079.00	0.125	5955.50	2	2 (6)	0 (0)	✓
R7	2958.00	0.063	2958.00	0	0 (0)	0 (0)	✓
R8	0.00	0.203	7860.00	3	3 (7)	0 (0)	✓
R9	3574.00	0.250	7384.00	6	3 (10)	3 (3)	
R10	2958.00	1.109	8907.00	12	12 (48)	0 (0)	✓
R11	1597.00	0.172	3499.50	1	1 (2)	0 (0)	✓
R12	0.00	0.203	7403.00	2	2 (5)	0 (0)	✓
R13	14068.00	0.453	17773.00	2	2 (7)	0 (0)	
R14	11898.00	0.531	12272.50	4	3 (14)	1 (3)	
R15	7091.50	1.188	13671.00	6	5 (20)	1 (2)	
R16	3733.00	0.407	4733.00	1	1 (4)	0 (0)	✓
R17	1170.00	0.375	5924.00	2	2 (6)	0 (0)	✓
R18	6369.00	0.703	15873.00	5	4 (14)	1 (2)	✓
R19	13067.00	0.422	14132.50	1	0 (0)	1 (2)	✓

Table 7.4: *LP solutions of the windy RANDOM instances.*

This is the data set which has given the best results. It seems related to the variability of the parameters involved. The fact of having a big standard deviation among the costs of these graphs seems to make easier the decisions made by the solver. Furthermore, the required times are very satisfactory, in all but two cases less than one second, which is closely related to the iteration numbers done, in all but one under 10.

The results for the 36 instances of the *GRID* set are depicted in Table 7.5.

name	Z_w	t_{Z_w}	Z_0	iter	con($\leq$)	coc ($\leq$)	
G0	0.00	0.032	1.00	2	2 (5)	0 (0)	✓
G1	4.00	0.047	4.50	3	3 (12)	0 (0)	
G2	3.00	0.047	4.00	2	2 (6)	0 (0)	✓
G3	9.00	0.062	9.00	0	0 (0)	0 (0)	✓
G4	8.00	0.031	9.00	2	2 (8)	0 (0)	
G5	9.50	0.031	10.50	2	1 (3)	1 (3)	
G6	16.00	0.125	17.00	4	2 (5)	2 (5)	
G7	9.00	0.063	10.00	3	2 (6)	1 (1)	✓
G8	10.50	0.078	12.00	4	2 (7)	2 (4)	
G9	11.00	0.281	13.00	7	6 (25)	1 (1)	
G10	10.50	0.343	12.50	4	3 (23)	1 (2)	
G11	16.00	0.250	16.67	2	2 (12)	0 (0)	
G12	33.50	1.000	35.50	6	4 (19)	2 (5)	
G13	28.25	0.907	30.00	6	5 (16)	1 (3)	
G14	32.67	0.688	33.50	3	2 (8)	1 (3)	
G15	47.00	1.375	47.50	6	3 (8)	3 (5)	
G16	38.22	1.469	40.00	9	6 (23)	3 (10)	
G17	44.00	0.844	44.50	3	2 (6)	1 (2)	✓
G18	28.00	1.016	29.00	4	4 (24)	0 (0)	
G19	30.25	2.765	33.00	10	9 (47)	1 (4)	
G20	30.00	3.109	32.50	7	6 (50)	1 (2)	✓
G21	61.50	3.500	64.00	7	4 (20)	3 (11)	
G22	63.00	4.609	65.50	7	4 (20)	3 (11)	
G23	66.00	3.906	66.00	4	2 (16)	2 (4)	
G24	95.50	6.687	96.00	6	2 (5)	4 (13)	
G25	80.00	6.312	82.50	9	6 (19)	3 (9)	
G26	93.00	6.140	94.50	8	6 (22)	2 (7)	
G27	46.00	6.297	48.00	7	6 (67)	1 (4)	
G28	60.50	9.437	62.00	9	7 (90)	2 (8)	
G29	50.94	16.406	52.50	16	14 (116)	2 (6)	
G30	96.25	32.859	97.75	11	8 (57)	3 (12)	
G31	101.00	47.015	102.00	15	13 (155)	2 (11)	
G32	108.00	13.062	108.50	7	5 (25)	2 (10)	
G33	154.50	14.032	155.00	2	0 (0)	2 (8)	
G34	145.67	19.047	146.00	4	1 (3)	3 (14)	
G35	150.00	10.656	150.00	1	0 (0)	1 (4)	

Table 7.5: *LP solutions of the windy GRID instances.*

Here we have 6 optimal solutions. In general, this data set has become especially experimental when translating it to the WCPARP. While the guidelines to construct these windy instances led to many zero costs, additional computational experiments indicated that a minimum positive threshold of one for the costs lead to an optimal value of zero for the vast majority of the instances, since the maximum value for profits is three. This fact should be taken into account for generating suitable values for the costs and profits for the computational experiments in future research.

In global, we can see that as expected, the required cpu time is strongly related to the number of variables. Even thought, upper bounds for the number of copies associated with each edge are not large, four as maximum, it is clear empirically that these new variables, representing the t-th traversals, are significantly time consuming.

Results are not as good as those obtained for the case of the CPARP. May be we should recall that these are preliminary results, and that in future research, new inequalities might be found for reinforcing the formulation. However, we maintain the number of optimal solutions at the end of the LP algorithm around 33%. Now a total of 41 out of the 118 instances were optimally solved by just solving the LP relaxation, thus giving integer solution values for the variables. Moreover, for solving all the instances a total of 1026.562 seconds was required, which means less than 20 minutes.

Summarizing the tables shown so far for all instances, we think that the overall results encourage to continue the research on this formulation, besides the task of creating some heuristic method and the final exact branch and cut algorithm for solving the WCPARP.

7.4.2 Solving the CPARP with the WCPARP algorithm

The results presented next are from the same data sets used for the CPARP replicating their costs for transforming them into WCPARP instances. Hence, for these problems $c_{uv} = c_{vu},\ \forall uv$.

Proceeding this way allowed us to contrast the obtained results against the old ones obtained with the CPARP algorithm for solving the LP relaxations. Tables 7.6-7.10 are presented here just for comparing the algorithms of the WCPARP versus the CPARP. In all of them, column headings are Z_w, the optimal solution value to the relaxation of the WCPARP formulation, followed by Z^*, the optimal value to the instance, since now we have them available. Then, under the heading *gap*, either a ✓ or the gap 100 $(Z_w - Z^*) / Z^*$ is displayed depending on whether or not the solution to the symmetric WCPARP instance is optimal. Recall that we know an instance to be optimal either because the LP solution is integer, or because the fractional value to the LP solution exceeds in less than one the optimal value of the corresponding solution in the CPARP. Next column, Z_r, has been repeated here from that of the corresponding column in Tables 6.8-6.12 of the Chapter 6. Finally, both times t_{Z_w} and t_{Z_r} corresponding to the required cpu times for solving the WCPARP and the CPARP linear relaxations respectively, are shown.

When the WCPARP is posed on a symmetric instance (that is, when $c_{uv} = c_{vu},\ \forall uv$), all dominance relations established for the primitive CPARP become

valid. Thus, in the solutions listed in Tables 7.6-7.10 there was no edge used more than twice, and always in these cases, in opposite directions.

name	Z_w	Z^*	gap	Z_r	t_{Z_w}	t_{Z_r}
ALBAIDAA	5351.00	5179	3.32	5179.00	21.094	5.890
ALBAIDAB	3536.00	3388	4.37	3394.29	46.047	8.672

Table 7.6: *WCPARP vs CPARP algorithms for ALBAIDA instances.*

For the ALBAIDA data set, the results are shown in Table 7.6. The gaps are reasonably small. The total amount of required time seems to strongly depend on the number of variables.

name	Z_w	Z^*	gap	Z_r	t_{Z_w}	t_{Z_r}
P01	3.00	3	✓	3.00	0.016	0.000
P02	28.00	26	7.69	28.00	0.078	0.047
P03	52.00	48	8.33	47.00	0.469	0.281
P04	33.00	31	6.45	33.00	0.141	0.047
P05	19.00	19	✓	19.00	0.078	0.031
P06	47.20	47	0.43	47.00	0.265	0.063
P07	73.00	73	✓	73.00	0.282	0.062
P08	74.50	72	3.47	72.00	0.078	0.032
P09	40.00	38	5.26	38.00	0.047	0.015
P10	35.00	35	✓	35.00	0.125	0.015
P11	9.00	9	✓	9.00	0.015	0.000
P12	8.00	8	✓	8.00	0.016	0.000
P13	3.00	3	✓	3.00	0.000	0.016
P14	97.80	92	6.30	98.17	1.141	0.234
P15	15.00	15	✓	15.00	0.219	0.063
P16	75.50	72	4.86	73.00	0.984	0.297
P17	23.00	20	15.00	22.00	0.203	0.047
P18	1.00	0		0.80	0.594	0.125
P19	49.00	49	✓	49.00	0.735	0.157
P20	194.50	182	6.87	189.67	4.594	1.140
P21	223.50	214	4.44	221.00	5.735	0.781
P22	399.50	395	1.14	398.00	4.360	0.797
P23	299.00	295	1.36	298.00	7.406	0.656
P24	173.00	171	1.17	173.00	1.875	0.375

Table 7.7: *WCPARP vs CPARP algorithms for CHRISTOFIDES instances.*

In Table 7.7 the optimal solution values with the gaps and their required times can be observed for the CHRISTOFIDES instances. There are 8 optimal (integer) solutions found when using the WCPARP, and, as in the case of the CPARP, one more data graph, Instance P18, whose optimality is proven at this point. Gaps are not so good, and in case of final results it would not be

acceptable. Note also like in the previous data set, that the required times for the WCPARP algorithm duplicate at least the times for the CPARP in all cases except for Instance P13 on which the windy algorithm runs faster than the undirected one. Also noticeable is the fact for Instance P14 the optimal value given by the WCPARP algorithm is a better upper bound for the integer optimal solution than that output by the CPARP. This is, for Instance P14, $Z_w < Z_r$.

name	Z_w	Z^*	gap	Z_r	t_{Z_w}	t_{Z_r}
D0	109.00	109	✓	109.00	0.000	0.032
D1	0.00	0	✓	0.00	0.015	0.031
D2	123.00	123	✓	123.00	0.031	0.047
D3	109.00	109	✓	109.00	0.031	0.062
D4	35.00	35	✓	35.00	0.000	0.031
D5	270.00	270	✓	270.00	0.016	0.032
D6	115.00	115	✓	115.00	0.015	0.046
D7	55.00	55	✓	55.00	0.032	0.063
D8	367.00	367	✓	367.00	0.015	0.062
D9	80.00	80	✓	80.00	0.047	0.312
D10	0.00	0	✓	0.00	0.078	0.421
D11	136.00	136	✓	136.00	0.235	0.782
D12	121.00	103	17.48	103.00	0.094	0.422
D13	559.00	496	12.70	496.00	0.313	0.328
D14	545.00	545	✓	545.00	0.328	1.422
D15	555.50	520	6.83	528.50	0.203	0.906
D16	840.50	783	7.34	801.00	0.328	0.453
D17	902.50	871	3.62	871.00	0.234	0.844
D18	286.00	256	11.72	256.00	0.844	2.453
D19	295.00	282	4.61	282.00	0.250	2.235
D20	180.00	180	✓	180.00	0.250	2.047
D21	777.50	768	1.24	768.00	1.266	4.844
D22	755.00	733	3.00	733.00	2.000	7.797
D23	772.00	724	6.63	738.00	1.109	3.718
D24	1295.50	1281	1.13	1281.00	0.516	2.859
D25	1047.50	1011	3.61	1012.60	1.765	5.938
D26	1366.50	1310	4.31	1310.00	0.797	5.391
D27	250.00	241	3.73	241.00	37.797	366.688
D28	497.50	483	3.00	483.00	5.828	37.296
D29	201.21	175	14.98	175.00	7.734	64.313
D30	958.31	883	8.53	933.00	9.485	66.187
D31	1222.50	1195	2.30	1195.80	5.719	29.921
D32	747.00	720	3.75	720.83	9.235	23.250
D33	1642.67	1550	5.98	1567.50	5.781	42.375
D34	1511.50	1436	5.26	1444.00	9.812	49.454
D35	1680.83	1579	6.45	1579.00	5.922	11.454

Table 7.8: *WCPARP vs CPARP algorithms for DEGREE instances.*

The results obtained for the DEGREE instances are depicted in Table 7.8. For this bench, the number of optimal solutions obtained with the WCPARP algorithm is 14.

Again the comparison between times leads to guess that the factor that cause a major impact is the number of variables. An important fact that has not been commented so far is the average difference between both solutions. Indeed, it is not small, since for some instances the ratio between the solutions with the WCPARP procedure and the CPARP one is over the 117%. This can not lead to other conclusion that the specific algorithm for the undirected case is still useful while no better results with the windy one be obtained. The reason for this better quality of the undirected procedure must reside in the fact that for the CPARP we have defined cuts on the x variables, forbidding hereby many edges to appear in the solutions.

In Table 7.9 the results of the WCPARP and the CPARP algorithm might be contrasted for the RANDOM data set, in which most of the instances have been optimally solved.

name	Z_w	Z^*	gap	Z_r	t_{Z_w}	t_{Z_r}
R0	0.00	0	✓	0.00	0.047	0.016
R1	236.00	236	✓	236.00	0.047	0.015
R2	0.00	0	✓	0.00	0.062	0.000
R3	8605.00	8605	✓	8605.00	0.047	0.016
R4	11206.50	11027	1.63	11027.00	0.031	0.016
R5	0.00	0	✓	0.00	0.094	0.016
R6	6681.00	6681	✓	6681.00	0.110	0.062
R7	3556.00	3556	✓	3556.00	0.078	0.016
R8	1946.50	1132	71.95	1132.00	0.172	0.063
R9	7544.50	7079	6.58	7079.00	0.219	0.078
R10	6025.00	6025	✓	6025.00	0.813	0.219
R11	2603.00	2603	✓	2603.00	0.172	0.031
R12	0.00	0	✓	0.00	0.203	0.016
R13	18398.00	18059	1.88	18059.00	0.547	0.250
R14	18176.00	16944	7.27	17023.75	0.406	0.250
R15	11858.00	10794	9.86	10794.00	1.047	0.375
R16	4544.00	4544	✓	4544.00	0.375	0.078
R17	1732.00	1732	✓	1732.00	0.359	0.063
R18	14055.00	14055	✓	14055.00	0.797	0.125
R19	17958.00	17958	✓	17958.00	0.563	0.172

Table 7.9: *WCPARP vs CPARP algorithms for RANDOM instances.*

As can be seen the number of optimally solved instances with the WCPARP procedure is 19, as in the case of the CPARP. Certainly the gap values continue to be not acceptable for all not optimally solved instances. Of course that this drawback should be addressed in future research.

Finally, for the GRID instances we can compare both algorithms in Table 7.10. With this data set 14 optimal solutions have been obtained using the

windy algorithm, whereas using the other one 21 were optimally solved.

name	Z_w	Z^*	gap	Z_r	t_{Z_w}	t_{Z_r}
G0	0.00	0	✓	0.00	0.016	0.000
G1	0.00	0	✓	0.00	0.047	0.015
G2	0.00	0	✓	0.00	0.031	0.016
G3	2.00	2	✓	2.00	0.094	0.047
G4	0.00	0	✓	0.00	0.063	0.031
G5	4.00	4	✓	4.00	0.031	0.016
G6	7.00	6	16.67	7.00	0.094	0.031
G7	1.00	1	✓	1.00	0.078	0.016
G8	4.00	4	✓	4.00	0.047	0.015
G9	1.50	1	50.00	1.00	0.281	0.079
G10	0.00	0	✓	0.00	0.422	0.141
G11	3.00	3	✓	3.00	0.297	0.093
G12	13.00	13	✓	13.00	0.734	0.438
G13	8.87	8	10.88	8.50	0.719	0.172
G14	13.00	13	✓	13.00	0.890	0.219
G15	23.00	23	✓	23.00	0.984	0.156
G16	15.94	15	6.27	15.50	1.938	0.312
G17	20.00	20	✓	20.00	0.860	0.203
G18	6.00	6	✓	6.00	1.578	0.375
G19	6.50	5	30.00	6.00	3.047	0.954
G20	9.00	9	✓	9.00	6.531	0.813
G21	29.00	29	✓	29.00	3.172	1.125
G22	30.00	30	✓	30.00	5.469	1.015
G23	29.00	29	✓	29.00	6.516	1.625
G24	51.00	50	2.00	50.00	6.141	1.578
G25	39.50	39	1.28	39.00	4.390	1.438
G26	51.00	51	✓	51.00	4.469	1.109
G27	13.00	13	✓	13.00	17.344	3.906
G28	22.62	21	7.71	22.57	20.328	11.766
G29	13.50	12	12.50	12.33	16.094	5.969
G30	43.50	41	6.10	41.70	19.969	10.187
G31	48.40	48	0.83	48.00	67.781	26.016
G32	49.50	49	1.02	49.33	20.218	5.031
G33	81.00	80	1.25	80.00	15.140	2.329
G34	77.33	76	1.75	76.00	18.766	6.453
G35	80.00	78	2.56	78.00	8.953	5.312

Table 7.10: *WCPARP vs CPARP algorithms for GRID instances.*

In global, among all the instances, the maximum number of copies for a given edge has been 4. In average, 1.5. This means that the number of variables defined, in average, has been the triple for the WCPARP formulation than in the CPARP. This factor seems to have a determinant impact when comparing the required times with the case of the undirected CPARP.

Also, the formulation should be reinforced somehow in order to avoid the gap values found so far for instances not optimally solved.

Chapter 8

Conclusions

In this dissertation we have studied the CPARP. This arc routing problem has two main characteristics. On the one hand, it is a Prize-collecting Arc Routing Problem, in which there is a profit function associated with links with demand, so that when a demand edge is serviced a profit is obtained. On the other hand, in terms of the components induced by the demand edges, it is required that if a demand edge is serviced, then all the demand edges in the same component are also serviced. As it has been explained the CPARP is closely related to other arc routing problems and, in particular, to the Rural Postman Problem.

In Chapter 3, we have proposed a transformation of the original CPARP into an equivalent problem defined on a complete graph, and we have studied several properties, based on which we have proposed a mathematical programming formulation for the CPARP. There are several elements specific of the proposed formulation. On the one hand, it exploits explicitly some dominance relations, which allow us to reduce the search space to optimal solutions with a specific structure. On the other hand, the formulation combines binary variables and continuous variables (within the interval $[0, 1]$), since we have proven that the variables that represent the second traversal of an edge can be defined as continuous ones but will take binary values in optimal solutions. The formulation contains two families of constraints of exponential size: restrictions that guarantee the connectivity with the depot of the serviced components, and cocircuit inequalities that ensure the even parity of the visited vertices. Therefore, for making it possible to use this formulation within the context of an LP solver iterative scheme, in Chapter 4 we have studied the separation problem for both types of inequalities, and in both cases we have seen that we can solve them exactly in polynomial time.

In Chapter 4 we have also proposed a branch-and-cut algorithm for obtaining optimal solutions to the CPARP. At each node of the search tree, the algorithm combines the iterative solution of the LP relaxation of he problem with a heuristic. The heuristic is a simple one which is based on rounding the obtained LP

solution. Despite its simplicity it has outperformed various types of heuristic that we have also studied in Chapter 5.

To assess the potential of the proposed formulation and the efficiency of the proposed algorithm in Chapter 6 we report on the results that we have obtained with a series of computational experiments. Since no benchmark instances were available for the CPARP we have adapted several families of well known arc routing instances from the literature. The obtained results indicate that in the vast majority of the cases the instances can be optimally solved with a small computational burden. It is quite remarkable that over 75% of the instances were already optimally solved at the root node of the search tree. Since the RPP is a particular case of the CPARP, a second series of experiments were oriented to evaluate the capacity of the proposed formulation for solving classical RPP instances. The obtained results are quite satisfactory although, as could be expected, these are not as good as the ones obtained with best existing RPP specific algorithms in the literature.

In Chapter 7 we have studied the windy version of the CPARP. For this problem we have proposed two different formulations, the first one based on general integer variables, and another one with binary variables, resulting from a transformation to a complete graph similar to that used for the CPARP. However, not all the properties that held for the CPARP when stated on the complete graph are also true for the WCPARP. In particular, it is no longer true that there exists an optimal solution in which no edge is used more than twice. As a consequence, for formulating the problem using only binary variables it is required to make several copies of the variables representing the traversal of a link in one direction. To this end, we have previously obtained an upper bound on the number of times that an edge can be traversed in a given direction in an optimal solution. The results of the preliminary computational experiments indicate the potential of the formulation, although it is clear that a thorough research on the WCPARP is one of the open lines for future research from this dissertation.

The Appendix describes the *filògrafus* which is a visual tool that has been implemented for working with graphs. It allows to edit graphs and to solve several graph optimization problems by means of a visual interface. In the context of this dissertation the *filògrafus* has been mainly used with CPARP problems. With respect to the solutions obtained with the iterative LP solver, the *filògrafus* is able to display the iterative process step by step. Related to the heuristic solutions, it is also used as a means of visualizing different options that the heuristic methods have parameterized. However, given that it is very versatile and user friendly, the *filògrafus* has a wide potential of applications, not only as a visual support tool for research activities, but also for teaching purposes.

From the research point of view, there are indeed several other aspects related to this dissertation that deserve further consideration. In the case of the WCPARP, one of them is the study of the formulation with the general integer variables and, in particular, the study of valid inequalities for this formulation.

The well-known $K-C$ inequalities can be adapted to be valid for the WCPARP, although an algorithm for separating them must be investigated. Also, given that cocircuit inequalities cannot be used with this formulation, the study of valid inequalities to guarantee the parity of the visited nodes deserves special attention for this formulation.

This dissertation also leaves the door open to the study of other problems. One of them is the classical RPP. We have seen that our CPARP formulation is able to solve the RPP instances quite efficiently. Given that it is already known how to formulate the RPP using only binary variables, it remains to be studied if there are any additional properties or dominance relations that hold when formulating the RPP on a complete graph. These could be incorporated thus leading to a stronger formulation, associated with the problem on the complete graph. The algorithm could compete with the up to date most efficient ones. One point that could contribute substantially to this would be to identify variables whose integrality could be relaxed and could be defined as continuous ones, similarly to the CPARP.

Finally, we must mention that there are indeed other types of prize-collecting arc routing problems that remain to be studied. Among them, probably the most challenging ones are the capacitated versions of the PRPP and the CPARP. None of these problems have been studied so far in the literature. The first step would be studying the properties of these two problems, and to see if it is possible to derive any additional property by formulating them on a complete graph. Along the line followed in this dissertation, it would be interesting to formulate these capacitated versions by means of binary variables and to exploit the potential of cocircuit inequalities. However, it is clear that the capacity constraints incorporate an additional difficulty to the problems and that it will be difficult to extend the obtained results. These are clearly avenues for future research.

Appendix A

The *filògrafus* Application

In order to test the hypotheses on the instances we have been working on, and looking forward to handling other graph instances in general, a software has been developed by the author. It has been called *filògrafus*: a visual application for operating with graphs. It consists of a workbench for graphs. Editing tools for elements (edges and vertices) are provided, as well as mechanisms for attributing the values involved in the problem. Even though initially the *filògrafus* had the purpose of working on several types of problems, the tools supplied for working on the CPARP have become its central objective. Other programs implemented as command lines programs are the ones whose results have been presented in Chapter 6. Along this chapter, these programs are referred to as *batch* programs.

The general requirement for the *filògrafus* is to be able to follow the resolution of a CPARP instance allowing the user to stop and analyze each step. While batch programs can do not much more that giving the solution values, within the *filògrafus*, a problem might be partially solved, analyzed, and solved completely or not. Solutions can be eliminated at any step, and the process restarted.

This interface application has been built in Microsoft VisualC++ language to run on a WindowsXP operating system platform. The command line conducted *batch* programs, without graphical interface and focused on solving the CPARP have been coded in standard C++ language, facing portability to other platforms.

Next, a brief description of the database used by the *filògrafus* is given. Then, data structures are commented explaining its hierarchical structure in layers. The main information stored at each layer and its main methods are described. There is a third section mentioning the policy of working with dynamic link libraries. Finally, a section tracing the main use case for solving a CPARP step by step is detailed.

A.1 Database

The application has knowledge about some routing problems, as the CPP, the RPP, the PRPP, and its speciality of course, the CPARP. These data structures are referred to as *models*. It might result convenient to say that having knowledge of a problem does not imply to know how to solve an instance of it. For solving a particular instance of a model a dynamic link library for that model is required.

In order to have the knowledge available, the *filògrafus* makes use of a simple database in which there is a list of *models* and a list of *attributes*. Also, relating both lists, there is a crossed relation, called *uses*. Hence, using Microsoft Access, the user might define new kinds of problems by associating problems with attributes. In each association the user establishes the domains for each attribute that uses her or his problem, and other information as the model name, an explanation (the *filògrafus* has a didactic spirit), a code, whether or not it requires binary variables, and also whether or not it is stated on a directed graph. Other machine oriented info is also stored for each *problem* as is the UID code for registering the corresponding dynamically link library (dll), as explained in Section A.3. The associated info for each attribute is its name, whether it is defined on the edges or on the vertices, and which data type implements it. The attributes also might exclude other attributes (e.g. in a maxflow problem there is an attribute associated with the vertices that is *sink* that excludes the attribute *source*). For each attribute a *readonly* flag is also stored (e.g. flows can not be given by the user).

A.2 Data Structures

The main program of the *filògrafus* uses a default data structure consisting of three different layers.

At the bottom layer there is the *kernel* object. This object is responsible of keeping the connectivity information among elements. Its main members represent the numerical implementation of the graph on which the problem is posed. That is, the matrix of connections (nodes×nodes) storing the costs of the edges, and the incidence matrix (edges×nodes). One more squared matrix (nodes×nodes) stores edge identifiers, that at this bottom level are integer indices. For a *kernel* object, subsets of edges are implemented as vectors of edge indices. At this first layer available methods are *dijkstra(i,j)* (for obtaining the shortest paths from a vertex i to another j as a vector of indices of edges, and the corresponding value), *min_ span_ tree()* (that also produces a vector of integer indices to edges as well as the value of the tree), *perfect_ matching()* (that returns the selection of edges that are the matching solution), and also *maxflow(i,j)*, (that interprets the costs matrix as capacities).

On the second level in the hierarchy there is the *graph* object class. It is responsible for the consistency among all the values of the graph and its domains. Additionally to kernel data, at this level there are two arrays. Vertices and edges. These arrays are collections of what in turn are data structures that maintain additional information of each elemental component of the graph. Data as name, position on the screen, an array of pairs noun-value for the element attributes, and other info that does not have relationship to the connectivity of the graph. A subset of edges for a *graph* is a slightly sophisticated data structure. Also at the layer of *graph* we have input/output operations available. This means that at this level methods need to know the domains for the values of the attributes in order to detect possible mistakes in the input data files. Therefore, it is in the *graph* level where all connections to the external data base are carried on. The topmost level data structure for general problems is *problem*.

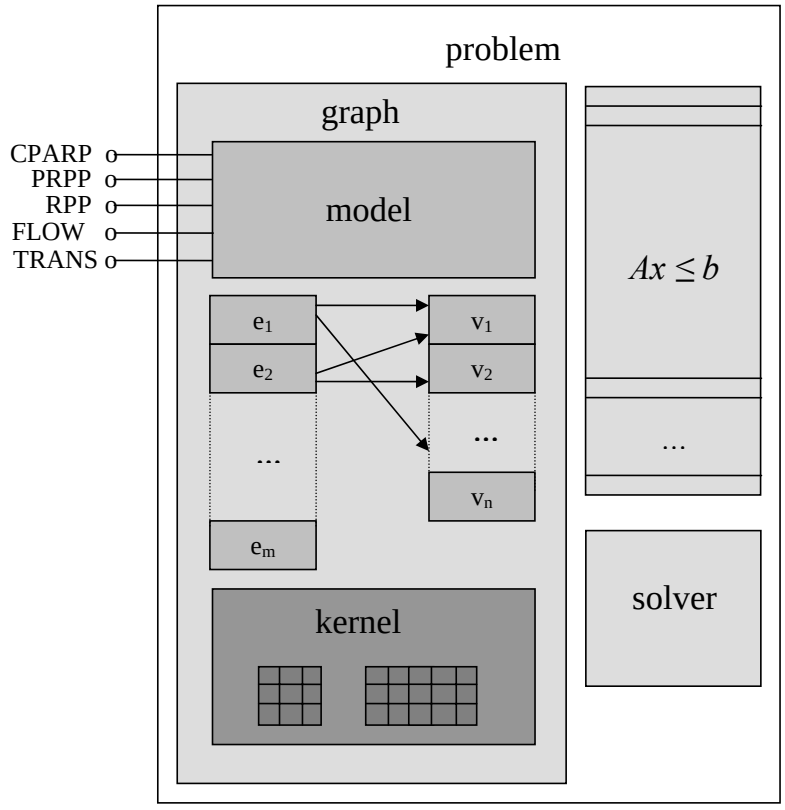

Figure A.1: *Hierarchical data structure used for the generic problem in the* filògrafus.

The responsibility of the interaction with the solver is assigned to this level. The solver used by the *filògrafus* is GLPK [91], conveniently adapted for this project. The *problem* data structure, in addition to all information provided by the *graph*, stores the matrix of inequalities of the problem. Observe that at this level only data structures can be provided, since operations for setting the

contents of this matrix (as the initial model or the separation procedures) are problem dependent, and therefore, are implemented in the particular library for that model.

A scheme of the data structure of the *filògrafus* can be seen in Figure A.1.

A.3 Dynamic Link Libraries

Some mechanism must be provided in order to set the specific features for a given problem. As said above, there is a global unique identifier (GUID) associated with each *model* in the database. When the user wishes to work on a new kind of problem, an executable module must be implemented and registered in the windows registry. Thus, the executable object is split into as many executable objects as *models* that the *filògrafus* can solve.

So *models* are objects that implement several interfaces. Examples of mandatory functions contained in these interfaces are *objective_function(), restrictions(), separation_of_inequalities()* or *addition_of_inequalities()*. All model-dependent characteristics of the problem must also be carried on these libraries. Therefore, the *filògrafus* itself does not know anything about demand edges or clusters.

So far, dynamic libraries exist for the CPARP, the PRPP, the RPP, the MAXFLOW and the TRANSPORTATION problems. Thus, these are the problems that currently the *filògrafus* is able to solve.

A.4 Workflow

For illustrating purposes, in this section it is described the interactive process for solving the LP relaxation of the Christofides instance P02, with the *filògrafus*. When the user starts by creating the instance this process is partitioned in three stages. However, for instances read from a file, the first stage, that consists of obtaining an *operable* graph is usually skipped (unless the graph in the file is disconnected). In order to differentiate among these stages, different keywords appear in the status bar at the bottom right of the main window.

stage 1 In the first stage the graph is being build. That is, it might be disconnected. It might have no defined depot, or it might have no demand edges. In any case, the instance is considered not *operable*. Therefore, when the graph is connected, has some demand edges and a depot vertex, the keyword *operable* appears on the status bar indicating so. As said above, this stage is usually skipped by the files of the workbench.

stage 2 These stage goes from having a connected graph with some demand edges and one depot vertex, to having a complete graph satisfying the triangular inequality and with all vertices with some demand edge in their cut set. When all these conditions are accomplished, then the graph is preprocessed. In fact, the word *preprocessat* indicates the end of the second stage.

stage 3 This is the final stage. It deals with solving the CPARP posed by the graph constructed in the previous stages. On the status bar, appears the last obtained solution value.

There are two user options for automatically carrying out the second and three stages. Thus, in order to solve step by step a given instance, we must uncheck options related to automatic preprocessing and solving. Also, the choice of solving step by step is assumed to be on.

A.4.1 Obtaining an operable graph

The *filògrafus* is able to read text files with the description of an edge in each line. Available files fulfil this format. They are ASCII files without extension. So, the *filògrafus* reads files without doing any previous transformation. To maintain the positions of the vertices in the graph, an auxiliary file is stored for each instance, with its positions. When opening a file for the first time, vertices appear in a default layout. By dragging them, the user can place vertices as desired, and they will be kept for future visualizations of the same graph if saved.

Once loaded, demand edges appear in bold. Profits and costs can be seen using the buttons on the main frame menu. Also, clusters can be selected and their properties shown, as in the screen shot displayed in Figure A.2.

Briefly, the options of the menu bar of Figure A.2 are, from left to right, *new graph*, *open*, *save* (that saves the positions of the vertices in the screen), *about*, (the red circle is an auxiliary button), and then comes the button with the legend $Ax < b$ for setting the main options for this model. Among them, now we have chosen that we do not want to preprocess the graph, also that we want to solve it step by step. Other options under this button are the kind of heuristic we would use now, or the column headings we want to appear in reports. After that, there is the button for calculating the minimum spanning tree. The algorithm used for this is parameterized in the dialog of graph properties not shown in this example. Next button calculates the minimum weight matching. At its right side the button ($\leq \triangleleft$) is checked, meaning that the instance already fulfils the triangular inequality. The button for constructing the complete graph follows with an icon of a graph in green. Note that this one is not checked since the graph is not complete as can be seen in the status bar.

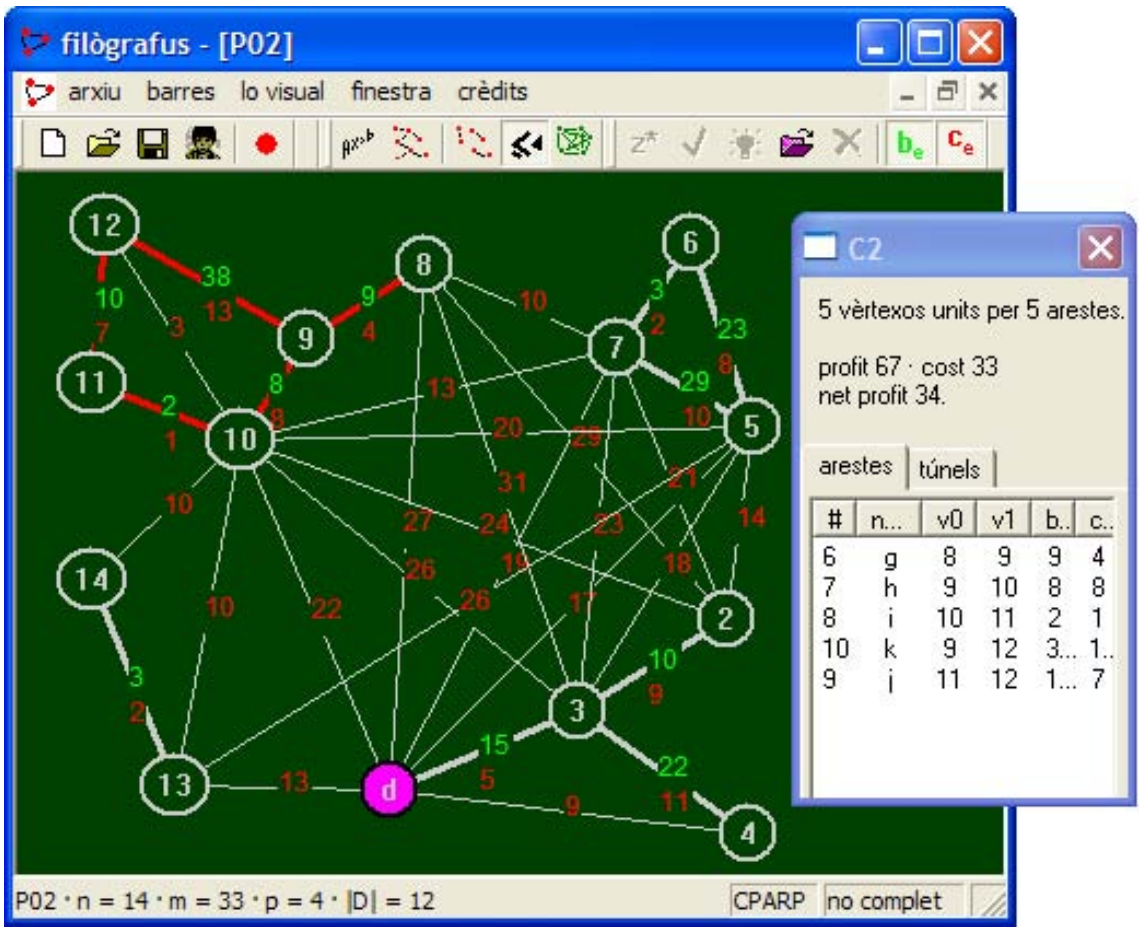

Figure A.2: *The instance P02 just loaded. Costs in red, profits in green for demand edges, in bold. The properties of cluster C_3 are shown in the dialog box.*

The following four buttons are related to the solutions: optimal (Z^*), relaxed (✓), heuristic (light bulb) and imported (folder in magenta). Observe that given that the graph is not preprocessed (since it is not complete), the first three are inhibited. However, at this point we already can import a solution file. This is shown in Figure A.3, which depicts a solution read from the file named *s.set*, as can be seen in the rightmost text of the status bar, with a solution value of 28.00. The next, is the button for eliminating solutions, now inhibited. Finally, the rightmost checked buttons are so because benefits and costs are shown, but they will not longer be in the example of this section. Observe the dialog box window with the properties of C_2, colored in red, on the right part.

One can see, on the bottom of the main frame, the status bar that shows the name of the instance and its sizes on the leftmost part. In this bar, it also appears the model name, CPARP. And then the status of the solving process. Given that we are working on a connected graph with some demand edges already defined, the graph has already skipped to stage 2. In Figures A.2, A.4

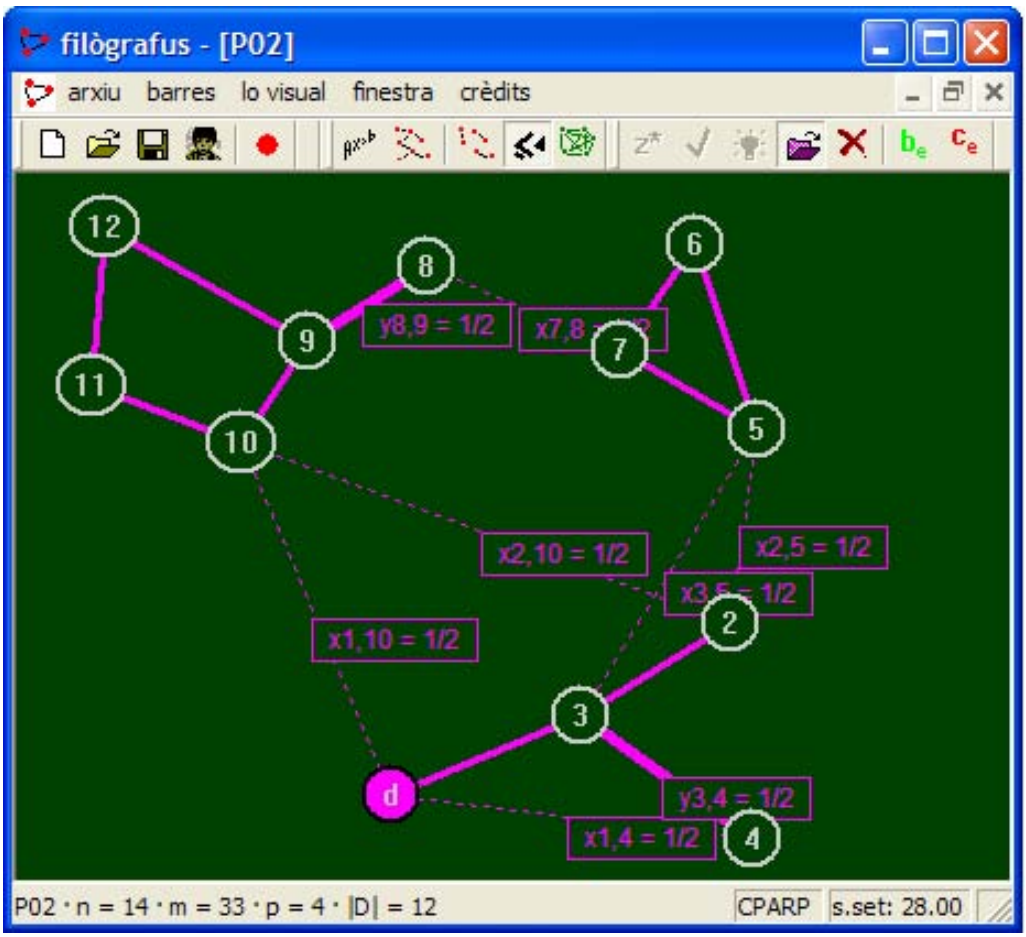

Figure A.3: *LP solution obtained by the batch programs and read from the* filògrafus.

and A.5 the status is *no complet* meaning that there is still some preprocessing that we have to do (making the graph complete) before achieving the status of *preprocessat*. Previous status could have been *disconnected* or even *isolated vertices*, *no depot*, and *no demand edges*. In Figure A.3 the status of the process has been overwritten by the filename and the value of the imported solution: *s.set: 28.00*.

Clicking again on the magenta colored folder or deleting the solution read with the button of the red cross, we turn back to our instance. Now the graph is only connected. When the problem status becomes *preprocessat* the buttons related to solving it become enabled. Even though the graph is not even complete, many operations are already available. The user has many actions that she or he can perform, in addition to completing the graph. Three examples of available actions before having a *preprocessed* problem are depicted in Figures A.3-A.5. The first example, in Figure A.3, has already been commented.

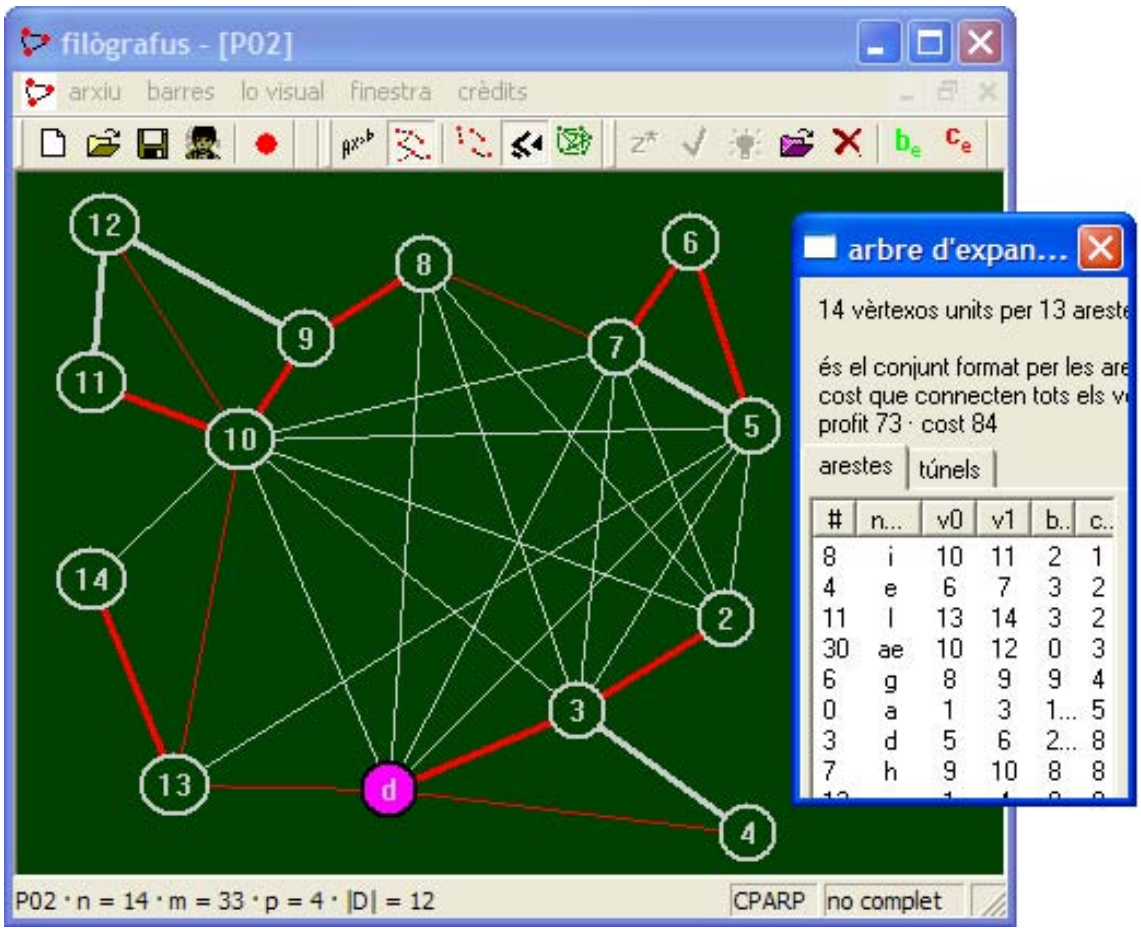

Figure A.4: *The minimum spanning tree.*

In Figure A.4 the user has calculated the minimum spanning tree. And the third example illustrates a tool for measuring paths. This tool, shown in Figure A.5, has resulted very useful for measuring a path between two vertices. For doing so, we just have to overdraw the path between the two vertices, that is enlightened as the user drags over a sequence of connected edges.

Other general utilities available to the user for analyzing the graph are not illustrated here to avoid extending too much this appendix. One of them is changing values of costs or profits. This turned out to be very interesting when comparing solutions, and also helps to figure out and make hypothesis when trying to design some good heuristic. Many buttons that have been hidden in this presentation for simplicity can be used for an extended variety of actions. In particular, any of the subsets described in Section 3.2.2 has a button for coloring edges belonging to it. Also the buttons for drawing odd and even vertices in different colors have been hidden.

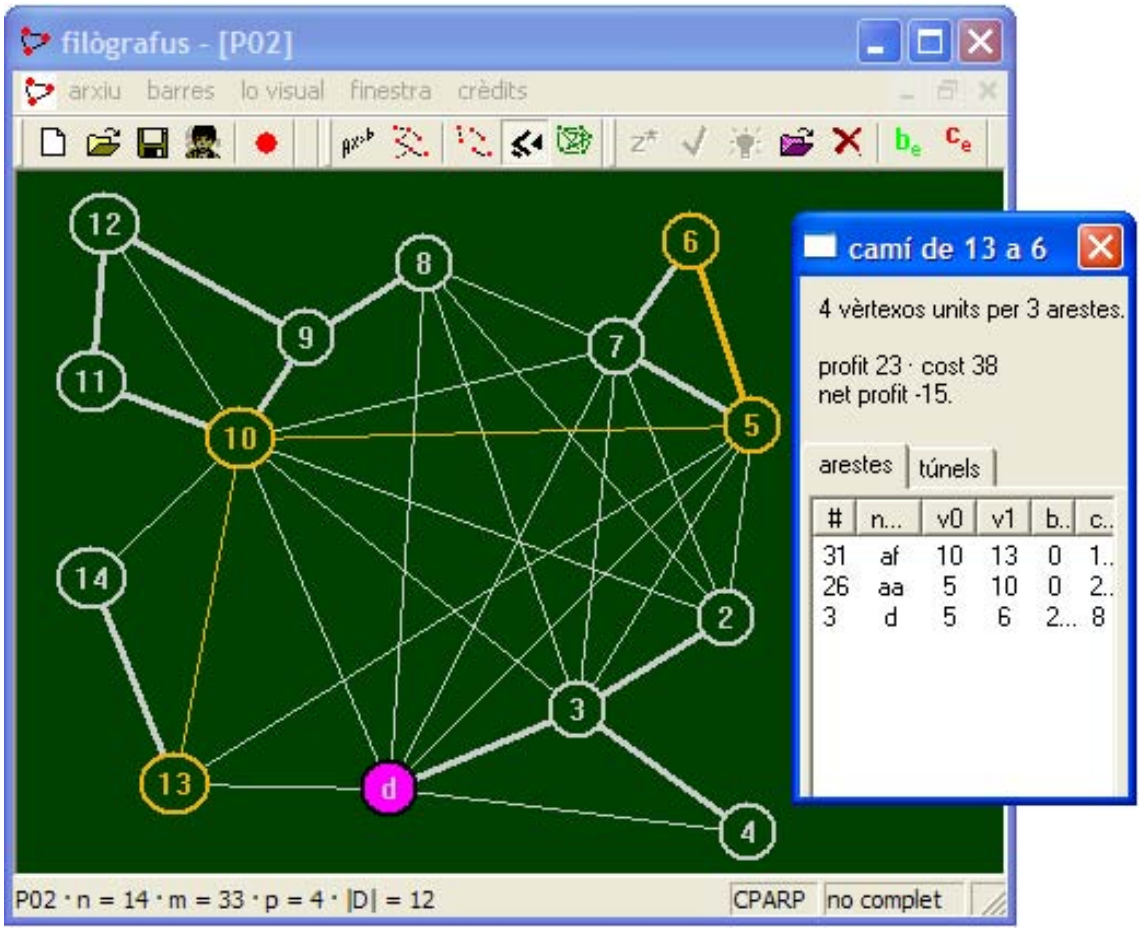

Figure A.5: *Measuring the attributes of a path.*

In fact, no contextual[1] menus are going to be shown in this example. With them, on the elements of the graph we would access to actions as modifying their values (e.g. the *filògrafus* can assign the role of the depot to any vertex by modifying its properties through its contextual menu), and by clicking the right button on the background we would be able to open the graph properties, the list of vertices, the list of edges, the matrix of connections among nodes, and the incidence matrix of the graph. Moreover, this contextual menu allows the user to see the constraints matrix when the problem is partially or completely solved. This possibility is interesting when, as in this example, the problem is solved step by step, since observing the evolution of this matrix is illustrative.

[1]In Windows operating system *contextual* menus are pop up menus that unwrap when the user click on the right button over an object

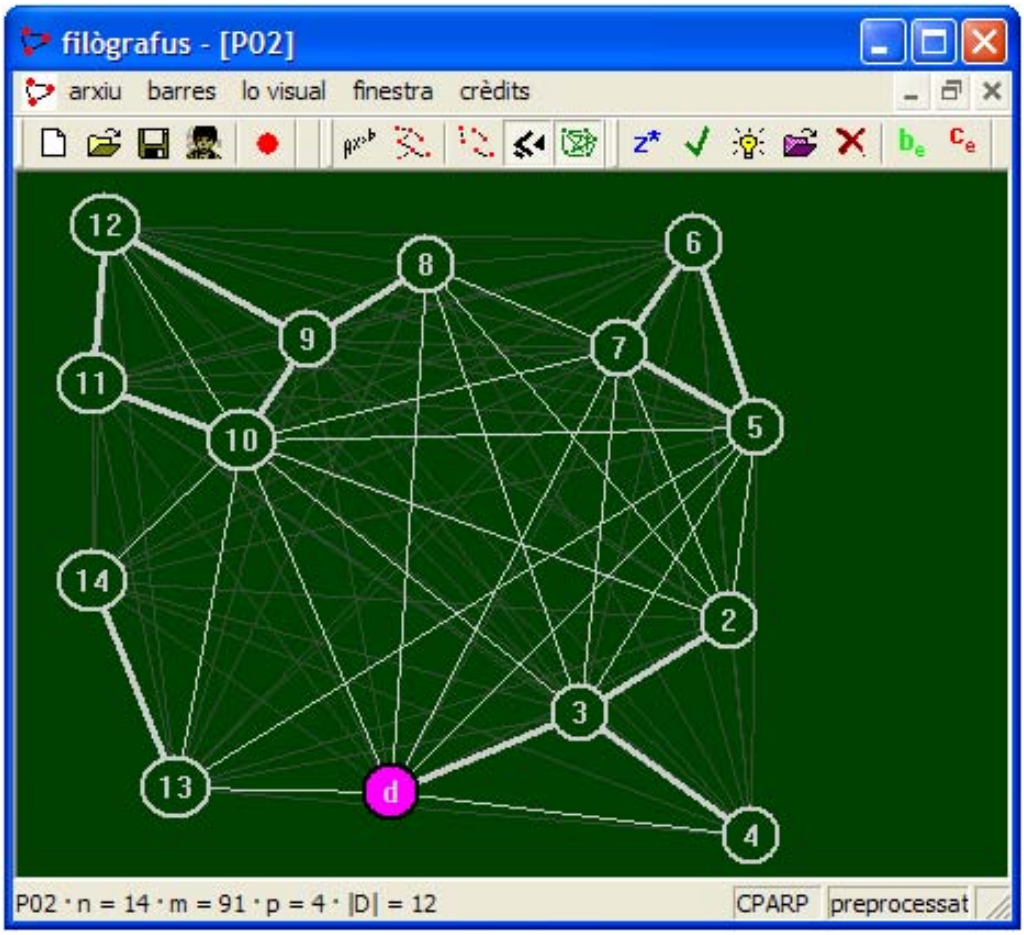

Figure A.6: *The completed graph.*

A.4.2 Obtaining a preprocessed graph

To continue solving the instance, next action to do is making the graph complete. By clicking on the button with the green graph we obtain the screen shot displayed in Figure A.6. This figure depicts all the edges of the graph that can be slightly seen. We can hide the edges that did not belong to the original graph by clicking again the same button. In fact, they will no longer be shown.

The reader might have noted that we have not talked about one of the preprocessing actions. The procedure for removing all vertices without demand in the original graph has not been dealt with, among other things because in Instance P02 such do not exist. So, the button for this action has not even been shown. Another action that must be done in the preprocessing is imposing the triangular inequality. Again, in this example, the property holds on the original instance, although in this case, the button has been maintained, and checked in all figures.

Once the graph has been transformed into a complete graph, several differences arise in the application frame window. In Figure A.6 the status bar can be observed, informing that all preprocessing has been done (*preprocessat*). Also, the number of edges in the right part of the status bar has been changed from 33, that was the original number of edges, to 91 = 14*13/2.

One more change has occurred in the menu bar. Now, all kind of solutions can be obtained. In particular, for this example, we are going to solve the LP relaxation.

A.4.3 Solving the instance

The user at this point can click on the green ✓ button of the menu. While solving one instance, the status bar informs of each step done, although with Instance P02 the user has no time to see these messages. Immediately, the screen becomes like the picture in Figure A.7.

Now, the dialog boxes that so far have not even been mentioned play a crucial role. Observing the popup window in Figure A.7 the user might closely follow the process. In its caption the user can see the notation *Zr* to recall that we are watching a solution to an LP relaxation. Then, iteration number between parenthesis, and the solution value.

Inside the dialog box window of Figure A.7, first observe that on the screen we have the last tab of this dialog box, entitled CPARP. Next on its left is entitled *arestes*[2]. Clicking on that tab would show a list of the edges in the solution. Other tabs not even shown are entitled *constraints* and *variables*. In particular, it has turned out to be very interesting to look at the constraints tab. It shows a list of all constraints already added to the formulation. When the user clicks on a constraint, 15 or less edges become enlightened, i.e., if the selected constraint involves less than 15 edges, all of them are emphasized, but when the constraint affects too many edges (so that if all were enlightened the user could see nothing), then just a sample of 15 of them are so.

Let us turn back to the CPARP tab of Figure A.7. At the top of the tab, there ia a message informing about the solution state. It is integer, however, disconnected. The number of connected components is also shown. There is also a list headed *grumolls*[3] with an item for each connected component of the solution containing its set of clusters. By clicking on an item, the corresponding set of clusters is enlightened. Below the list appears the button that allows the user to continue solving. The text on it is a plus symbol, meaning that the *filògrafus* is going to add inequalities, and the set of clusters that violates the cut in question. So, we would click on the button labelled + 2 i 3, obtaining

[2] *arestes* is the to translation of *edges* to Catalan
[3] *grumoll* has been chosen to translate *cluster* to Catalan

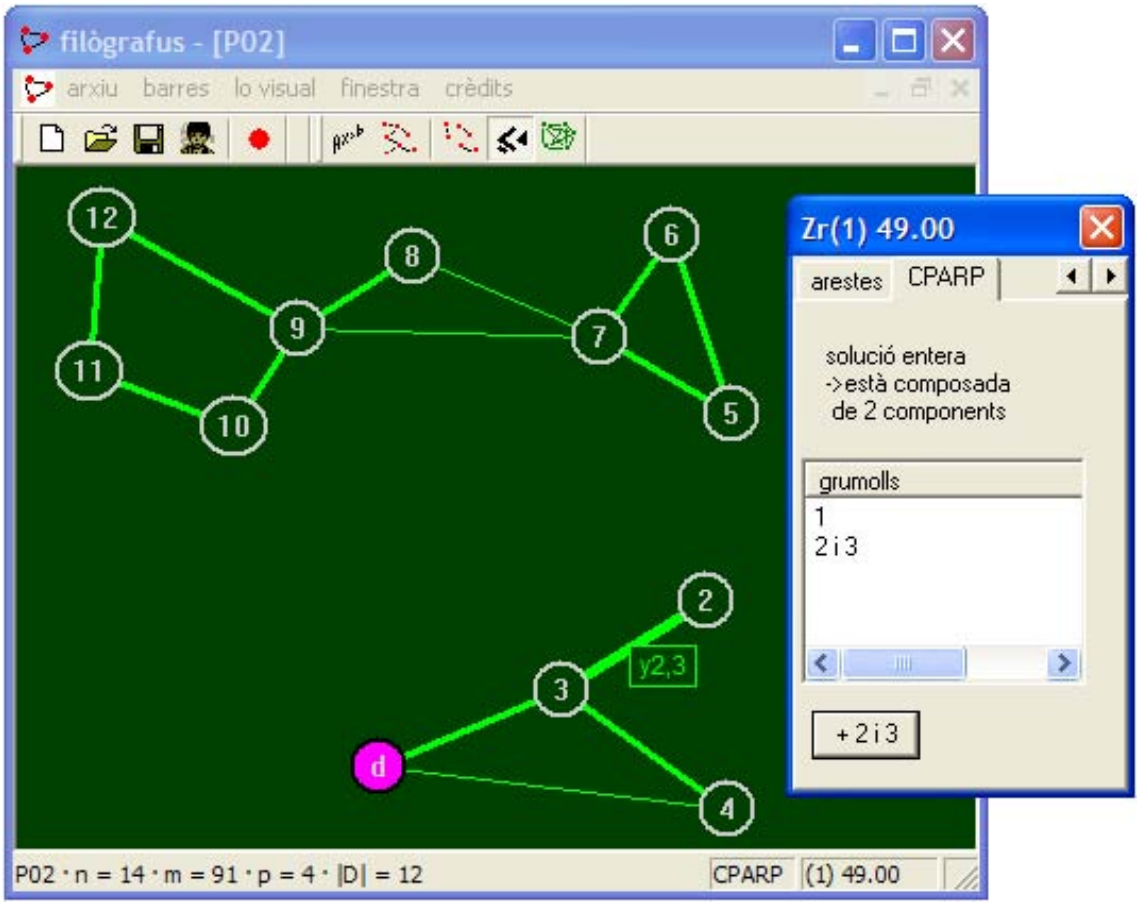

Figure A.7: *Solution of the first iteration.*

the result of Figure A.8.

In Figure A.8 the result of the second iteration is depicted. As can be seen, cluster C_3 appears in it, although only with $z_3 = 1/2$. This has been the way that the inequality just added has been satisfied. Now the solution is no longer integer. At least it is already connected. Nevertheless, it is not *enough* connected. With the separation procedure for connectivity inequalities described in Algorithm 4.3, the *filògrafus* found that the maxflow that can be sent from C_0 to any cluster of the set of $\{C_1, C_2, C_3\}$, that is equal to 1.0, is smaller than 2*max$\{z_1, z_2, z_3\} = 2 * 1.0$. Observe also that in the dialog box of Figure A.8 there is a text saying how many clusters resulted in the sink segment of the maxflow solution. There is also the cluster that gave the violated cut C_1, the maximum z_k variable value left in the sink segment of the solution (that in this case coincides with variable z_1), and the maxflow that can be sent to the cluster from the depot, 1. In this example, the resulting text is $C1(Z1 = 1)$ *amb flux* 1.

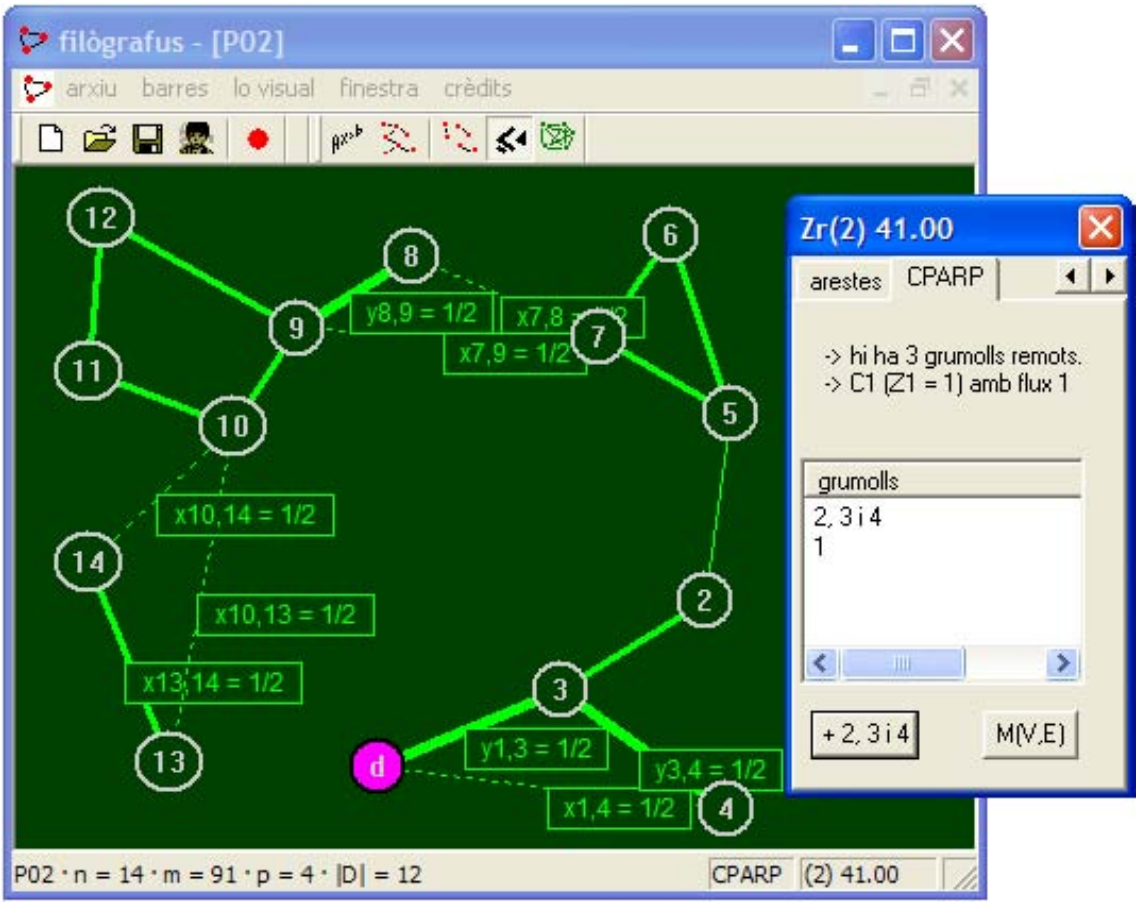

Figure A.8: *Solution of the second iteration.*

Before proceeding to solve the instance, we can get into the detailed process of the maxflow problem just solved. Observe that in the dialog box of Figure A.8, next to the button that would allow to continue solving, there is another button labelled $M(V, E)$. By clicking on it we open the MAXFLOW problem posed in the separation procedure. Therefore, the aspect of the *filògrafus* changes considerably. The screen turns out to look like the one shown in Figure A.9. The *filògrafus* has changed its model to MAXFLOW. Also in the status bar, the name of the current model has been set to MAXFLOW. In the MAXFLOW model the source vertex is colored in purple, whereas the sink one, in yellow.

In this way we can reproduce the Ford-Fulkerson algorithm implemented also inside the *filògrafus*. Batch programs use a static library. Observe that in Figure A.9 the name of the problem is an internally defined string. Also, the buttons of the menu bar have changed. This is the menu bar corresponding to the MAXFLOW model of the *filògrafus*. Only two new buttons need to be explained. Starting from the left, the second one that is inhibited in Figure A.9, shows the flow values (when solved) in cyan, and the third, the button that

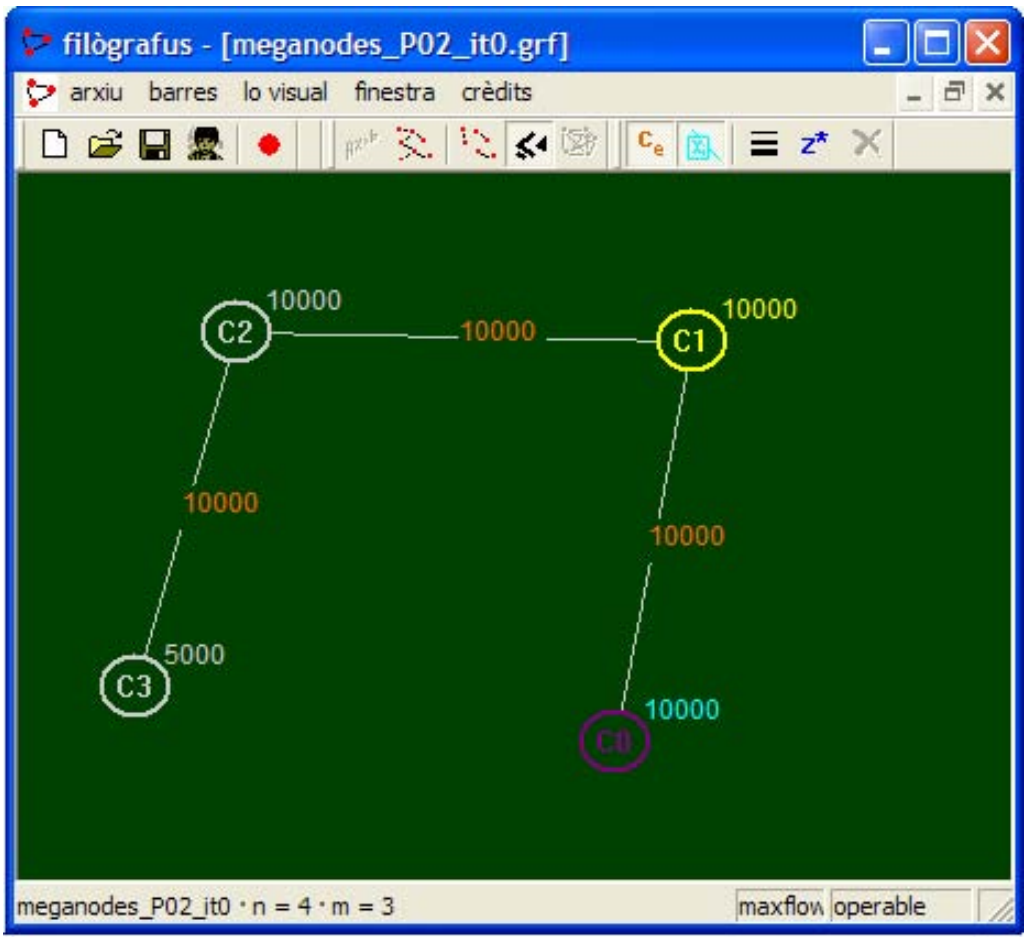

Figure A.9: *Graph M_s of the Algorithm 4.3.*

should interest to the user, with three horizontal lines in downward increasing thickness. This button is used to solve the maxflow problem. With respect to the graph elements, we have the edges capacities in orange, whose values correspond to that of Algorithm 4.3 scaled in order to work with integer capacities. There is also a numerical attribute associated with each vertex (the default MAXFLOW model has not numerical attributes associated with the vertices). The *filògrafus* does not care if we want to add some attribute to a model, provided that it does not affect the procedures for solving the instances of that model. In fact, the user can add and remove attributes associated with the elements in the graph properties dialog box already mentioned. The numbers close to each vertex in Figure A.9 are the values of the corresponding variables z_k (also scaled) in the solution of Figure A.8. They have no impact on the solution to the maxflow problem.

At this point the user can solve iteratively the maxflow problem by clicking the third button on the right of the menu bar. At each iteration an increment

of flow is achieved.

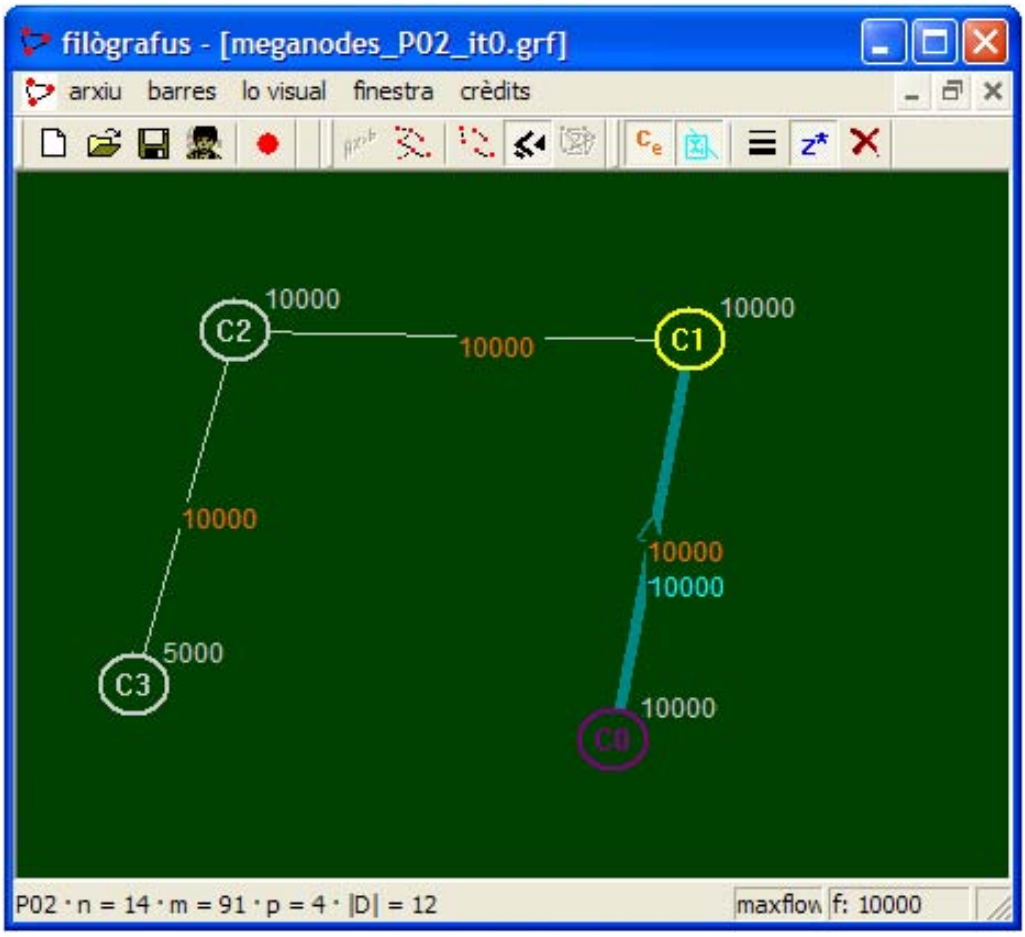

Figure A.10: *Solution to the maxflow problem of Figure A.9.*

Figure A.10 depicts the solution obtained at the first iteration of the maxflow problem. As can be seen, edge $\{C_0, C_1\}$ is thicker, blue, and with a small triangle indicating the flow direction in the solution. Also the value of this flow appears in cyan.

Once analyzed the solution of the maxflow problem, we simply close this window, and the CPARP on which we are working on appears again on the screen. Now it is time to add the connectivity inequalities found by clicking on the button labelled + 2, 3 i 4.

The result once added the last connectivity cut is depicted in Figure A.11. The dialog box indicates the current state. That is, there has been found no

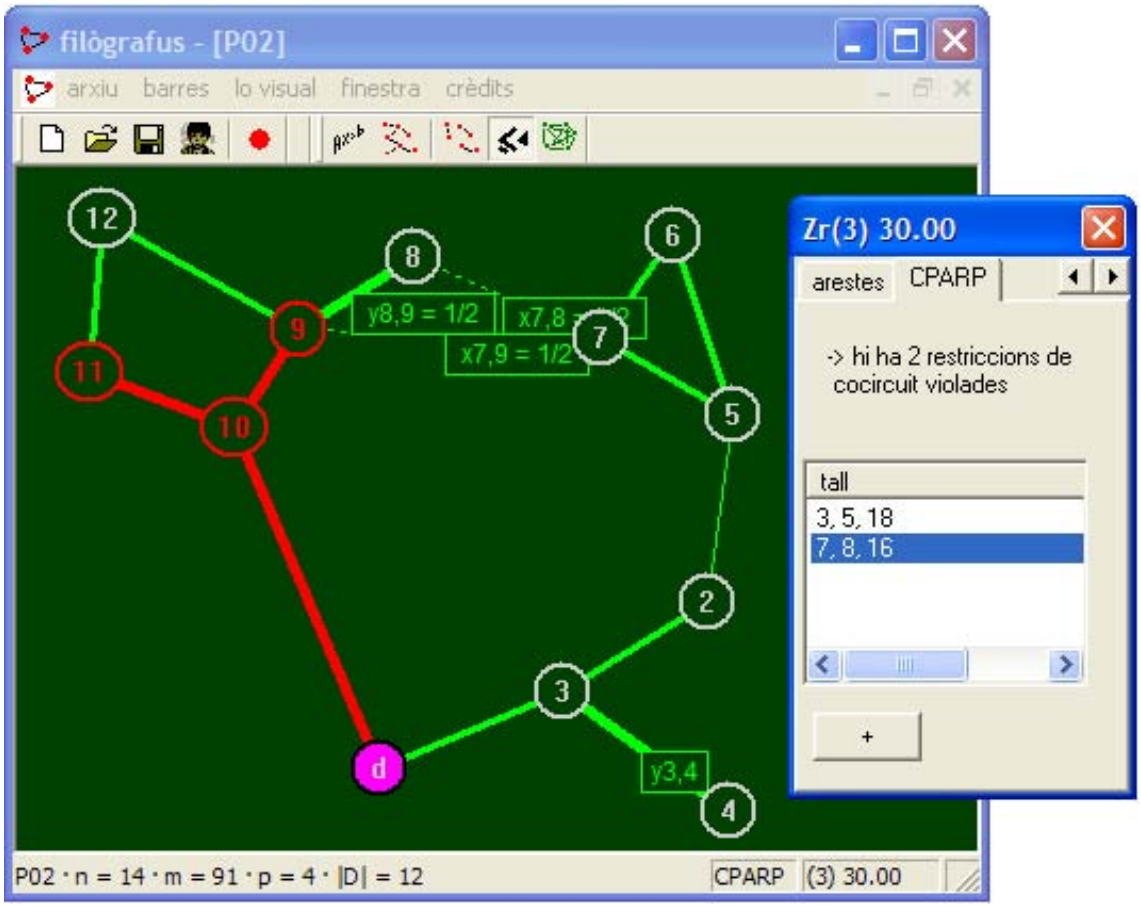

Figure A.11: *Solution of the third iteration.*

more violated connectivity inequalities, and therefore the separation procedure has proceeded by finding out a new cocircuit inequality to be added to the LP problem. In fact, there have been found 2. In Figure A.11, the list is labelled with the word *tall*[4]. Thus, items of this list are the edge indices of the cut sets $F \cup L$ of Algorithm 4.6. By clicking on one of them, the corresponding edges get colored in red, and thicker. Since the second item is selected, edges corresponding to this cut, {d,10}, {9,10} and {10,11} are marked in red in Figure A.11.

At the bottom of the dialog box window, the button labelled + can be clicked to add all cocircuit cuts of the list at once. Now the user can continue solving the problem by clicking this button. The obtained result is displayed in Figure A.12.

The solution of Figure A.12 satisfies all constraints of the formulation of

[4] *cut* in Catalan

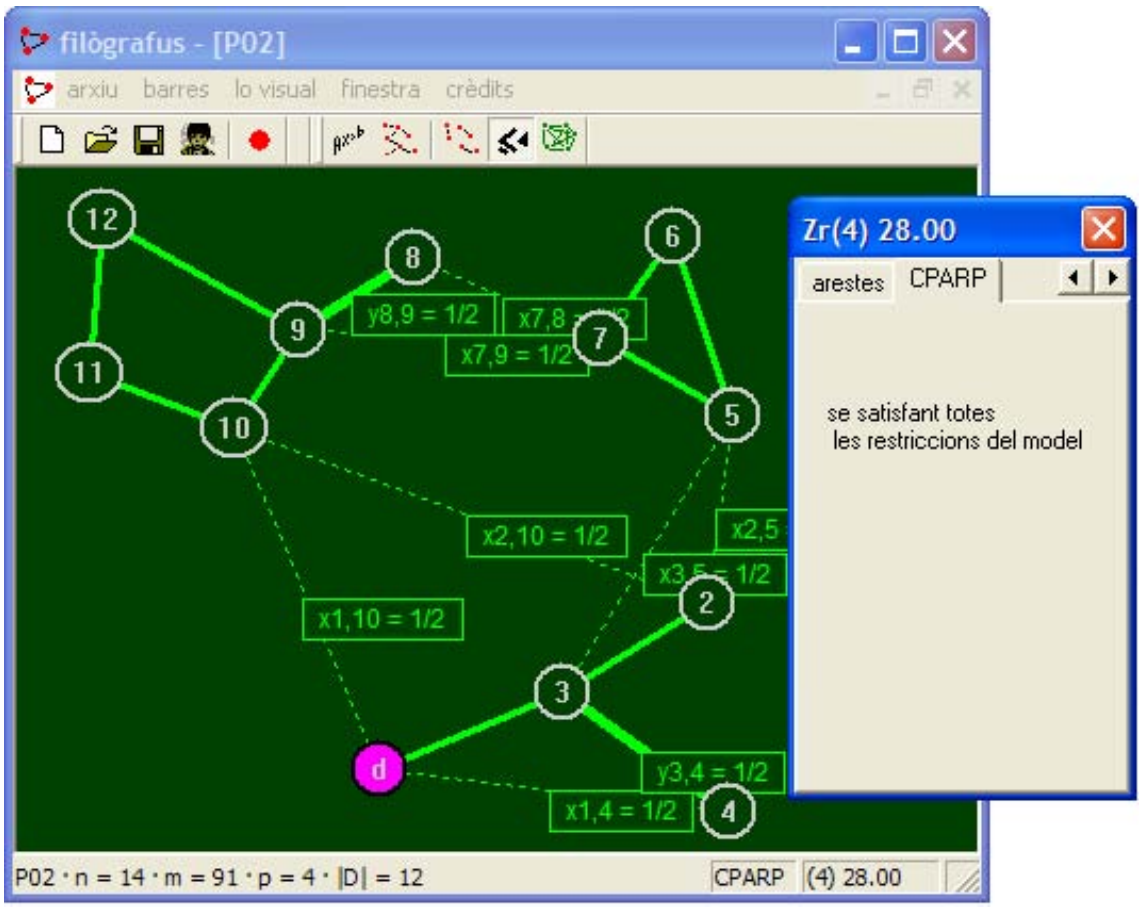

Figure A.12: *Solution of the fourth iteration.*

Subsection 4.1.3, the connectivity cuts of Expression 3.11, and all cocircuit cuts that the heuristic separation algorithm shown in Algorithm 4.6 has been able to find. Therefore, Algorithm 4.1 can not do anything else to improve the current solution value.

In Figure A.12, the optimal solution to our formulation is shown. However, it is unfeasible given that it has fractional values. Thus, the value of 28.00 is the upper bound that we obtain at the root node of the exploration tree. Observe also that the solution obtained using the *filògrafus*, in Figure A.12 coincides exactly with that calculated by the batch program, shown in Figure A.3.

Agraïments

Aquesta tesi ha estar parcialment financiada pel projecte del Ministerio de Educación y Ciencia OPTIMOS: Optimización para la Movilidad Sostenible (MTM2006-14961-C05-01).

Abans que res, i tal com resa en les primeres planes, m'agradaria dedicar aquest treball a la meva germana Marta Franquesa, que fins no fa massa m'ajudava en tantes coses. Ella em va fer comprendre què era la continuïtat,

$$\forall\ \epsilon\ >\ 0,\ \exists\ \delta\ (>\ 0)\ |\ |x-a|\ <\ \delta\ \rightarrow\ |f(x)-f(a)|\ <\ \epsilon.$$

Llavors, m'agradaria manifestar el deute moral que aquest projecte ha significat envers la meva directora Elena Fernández, que m'ha fet passar cinc anys dels més satisfactoris de la meva vida. També al codirector Julián Aráoz pels seus punts de vista tan savis. Molt enriquidor ha estat el contacte amb Ángel Corberán i José María Sanchís, especialment en l'últim capítol. Óscar Meza, desde lluny, per les dades i el seguiment que n'ha fet. I Maria Albareda per les seves puntualitzacions. Potser hauria de fer referència a tot el departament d'EIO de la UPC, però especialment m'agradaria mencionar a Jaume Barceló pel seu coneixement profund de tantes situacions, Jordi Castro pels seus seminaris tan interessants, i Ramon Nonell per la seva paciència a l'hora de fer-me comprendre les sigma àlgebres i els borelians, que no vaig entendre. Al Toni Font per la seva destresa amb els usuaris i les contrassenyes, i als companys Diana Cobos, Matteo Tesser, i Marcelo Smrekar, per abraçar-me a l'hora d'agafar el tren.

M'agradaria també donar les gràcies als companys del departament d'LSI de la UPC, Conrado Martínez, Maria Teresa Abad, Gabriel Valiente, Amàlia Duch, Edelmira Passarella i Cristina Zoltán. Treballant amb tots ells he après moltes coses, sobretot de la complexitat computacional, que també he utilitzat en aquest projecte.

Als meus companys del departament de MAiA de la UB, Anna Puig, Jesús Cerquides, Maite López, Josep Maria Bañeras, Oriol Pujol, Jordi Campos, Eloi Puertas, Daniel del Rio, Maria Salamó, Àngela Fàbregues, Petia Radeva, Jordi

Vitrià, Inma Rodríquez, Santi Ontañón, David Massip i altra gent del grup de recerca de Visualització de Volums i Intel.ligència Artificial (WAI). Ells m'han ajudat en les presentacions plantejant-me els seus dubtes.

Agrair tant com es pugui el suport moral que ha exercit el meu company Leo Colmena, sense qui tot això no tindria cap sentit. I altres companys de viatge com en Vicens Palà, d'un coneixement i una sabiduria extraordinàries, l'Albert Claret, per les seves lliçons sobre l'administració de l'espai en les interfícies, la Isabel Darnell que em controlava els cursos i em parlava dels números narcisos, en Gepeto per mostrar-me la complexitat a l'hora d'analitzar interaccions, l'Oriol Ferrer per il.luminar altres perspectives i l'Enric Santamaria per dibuixar-les. La Jurdina Conill amb qui compartia les tonades i els ritmes de totes les freqüències, en Carles Romeu i la Izaskun, com les agulles d'un rellotge budista, i en Joan Teixidó, en Joan Claret i l'Ester Sancho per ajudar-me a airejar quan m'encaparrava excessivament. La Marina Navarro per fer-me de mama i el José María Sánchez per fer-me de fill. En Josep Pi que em deia que tot té solució, i en Francesc Arnán per com s'omplen els espais buits. I l'Ángel Bustos i Mohammed Hammouda pels moments més difícils.

I també a les meves altres germanes Mercè per ensenyar-me a lluitar, Núria per allò de jugar escacs, i Eulàlia per la teoria de conjunts i al meu germà Jaume per ser el meu germà. Als amics Isidro Alonso, Jordi Roig, Albert Gonzàlez i Ferran Sanàbria, i als de la seva corda Sara, Laura, Neus, Alba, Pere, Enric, Ferran, Paula, Júlia, Martina, Carles i Anna. I encara, a la Maria, Blandina, Fina i Cesca Niubó, que també han estat amb mi sempre que m'ha fet falta.

Moltes gràcies.

Sant Pere de Ribes, a 1 de setembre de 2008

List of Figures

List of Tables

Bibliography

[1] D. Ahr and G. Reinelt. New heuristics and lower bounds for the min-max k-chinese postman problem. In R. Möring and R. Raman, editors, *Algorithms - ESA 2002, 10th Annual European Symposium. Proceedings*, volume 2461 of *Lecture Notes in Computer Science*, pages 64–74. Springer, Rome, Italy, 2002.

[2] M. Dell Amico, F. Maffioli, and P.Värbrand. On prize-collecting tours and the asymmetric traveling salesman problem. *International Transactions on Operations Research*, 2(3):297–308, 1995.

[3] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook. *The Traveling Salesman Problem (A Computational Study)*. Princeton University Press, 2006.

[4] J. Aráoz, W. Cunningham, J. Edmonds, and J. Green-Krotki. Reductions to 1–matching polyhedra. *Networks*, 13:455–473, 1983.

[5] J. Aráoz, E. Fernández, and C. Franquesa. The clustered price-collecting arc routing problem. *Graph Optimization Meeting GOM2008, Saint-Maximim La Sainte Baume, Agosto 2008*, 2008.

[6] J. Aráoz, E. Fernández, and C. Franquesa. The clustered prize-collecting arc routing problem. *Transportation Science*, -conditionally accepted-, 2008.

[7] J. Aráoz, E. Fernández, and C. Franquesa. The exact solution to the clustered price-collecting arc routing problem. *Conference of the International Federation of Operation Research Societies IFORS 2008, Sandton, Sudafrica, Julio 2008*, 2008.

[8] J. Aráoz, E. Fernández, C. Franquesa, and O. Meza. Prize-collecting arc routing problems and extensions. *ODYSSEUS*, 3 Mayo, 2006.

[9] J. Aráoz, E. Fernández, and O. Meza. A simple exact separation algorithm for 2-matching inequalities. *Research Report DR-2007/13, EIO Departament, Technical University of Catalonia (Spain). http://www-eio.upc.es/∼elena/Reports/DR200713.pdf.*, 2007.

[10] J. Aráoz, E. Fernández, and O. Meza. Solving the price-collecting rural postman problem. To appear in *European Journal of Operational Research*, 2008.

[11] J. Aráoz, E. Fernández, and C. Zoltán. The privatized rural postman problem. *Computers and Operations Research*, 33:3432–3449, 2006.

[12] A. Assad, B. L. Golden, and W. L. Pearn. The capacitated chinese postman problem: Lower bounds and solvable cases. *American Journal of Mathematics and Management Science*, 7:63–88, 1987.

[13] A. A. Assad and B. L. Golden. *Arc Routing Methods and Applications*, volume 8. Handbooks in Operations Research and Management Science, 1995.

[14] E. Balas. The prize collecting traveling salesman problem. *Networks*, 19(6):621–636, 1989.

[15] E. Balas. Price collecting traveling salesman problem and its applications. In G. Gutin and A. Punnen, editors, *The Traveling Salesman Problem and Its Variations*. Kluwer Academic Publishers, 2002.

[16] M. O. Ball and M. J. Magazine. Sequencing of insertions in printed circuit board assembly. *Operations Research*, 36(2):192–201, 1988.

[17] F. Barahona and M. Grötschel. On the cycle polytope of a binary matroid. *J. Comb. Theory*, 40:40–62, 1986.

[18] J.M. Belenguer and E. Benavent. The capacitated arc routing problem: valid inequalities and facets. *Computational Optimization and Applications*, 10:165–187, 1998.

[19] E. L. Beltrami and L. D. Bodin. Networks and vehicle routing for municipal waste collection. *Networks*, 4:65–94, 1974.

[20] E. Benavent, V. Campos, N. Christofides, A. Corberán, and E. Mota. An optimal method for the mixed postman problem. In Thoft-Christiansen, editor, *System Modeling and Optimization*. Springer.

[21] E. Benavent, V. Campos, A. Corberán, and E. Mota. Análisis de heurísticos para el problema del cartero rural. *Trabajos de Estadistica e Investigacion Operativa*, 36(2):27–38, 1985.

[22] E. Benavent, A. Carrotta, A. Corberán, J.M. Sanchis, and D. Vigo. Lower bounds and heuristics for the windy rural postman problem. *European Journal of Operational Research*, 176:855–869, 2007.

[23] E. Benavent, A. Corberán, E. Piñana, I. Plana, and J.M. Sanchis. New heuristics for the windy rural postman problem. *Computers and Operations Research*, 32:3111–3128, 2005.

[24] E. Benavent, A. Corberán, J.M. Sanchis, and I. Plana. Min-max k-vehicles windy rural postman problem. *Technical Report TR09-2007.*, 2007.

[25] L.D. Bodin and S.J. Kursh. A computer–assisted system for the routing and scheduling of street sweepers. *Operational Research*, 26:525–537, 1978.

[26] P. Brucker. The chinese postman problem for the mixed graph. *Lecture Notes in Computer Science*, 100:354–366, 1891.

[27] A. Caprara and M. Fischetti. Branch-and-cut algorithms. In *Annotated Bibliographies in Combinatorial Optimization*, page 45Ũ64. M. DellŠAmico, F. Maffioli, and S. Martello, Wiley, New York, 1997.

[28] N. Christofides. The optimum traversal of a graph. *Omega*, 1:719–732, 1973.

[29] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. *Report 388. GSIA. Carnegie-Mellon University*, 1976.

[30] N. Christofides, V. Campos, A. Corberán, and E. Mota. An algorithm for the rural postman problem. *Imperial College Report IC.O.R.*, 81.5, 1981.

[31] N. Christofides, V. Campos, A. Corberán, and E. Mota. An algorithm for the rural postman problem on a directed graph. *Mathematical Programming Study*, 26:155–166, 1986.

[32] G. Clark and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12:568–581, 1964.

[33] A. Corberán, A. Letchford, and J. M. Sanchis. A cutting plane algorithm for the general routing problem. *Mathematical Programming*, 90:291–316, 2001.

[34] A. Corberán, R. Marti, and A. Romero. A tabu search algorithm for the mixed rural postman problem. *Computers and Operations Research*, 27:183–203, 2000.

[35] A. Corberán, G. Mejía, and J.M. Sanchis. New results on the mixed general routing problem. *Operations Research*, 53:363–376, 2005.

[36] A. Corberán, E. Mota, and J.M. Sanchis. A comparison of two different formulations for arc routing problems on mixed graphs. *Computers and Operations Research*, 33:3384–3402, 2006.

[37] A. Corberán, I. Plana, and J.M. Sanchis. Zigzag inequalities: A new class of facet inducing inequalities for arc routing problems. *Mathematical Programming*, 108:79–96, 2006.

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

[39] A. Corberán, I. Plana, and J.M. Sanchis. The windy general routing polyhedron: A global view of many arc routing polyhedra. *SIAM J. on Discrete Mathematics*, 22:606, 2008.

[40] A. Corberán and J. M. Sanchis. A polyhedral approach to the rural postman problem. *European Journal Operation Research*, 79:95–114, 1994.

[41] A. Corberán and J.M. Sanchis. The general routing problem polyhedron. facets from the rpp and gtsp polyhedra. *European Journal of Operational Research*, 108:538–550, 1998.

[42] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[43] G. Cornuèjols, J. Fonlupt, and D. Naddef. The traveling salesman problem on a graph and some related integer polyhedra. *Mathematical Programming*, 33:1–27, 1985.

[44] R. Deitch and S. P. Ladany. The one-period bus routing problem: Solved by an effective heuristic for the orienteering tour problem and improvement algorithm. *Operation Research*, 127(1):69–77, 2000.

[45] M. Dror. *Arc Routing: Theory, Solutions and Applications*. Kluwer Academic Publishers, 2000.

[46] M. Dror and A. Langevin. A generalized traveling salesman problem approach to the directed clustered rural postman problem. *Transportation Science*, 31:187–192, 1997.

[47] J. Edmonds. Maximum matching and a polyhedron with 0,1-vertices. *Journal of Research of the National Bureau of Standards*, 69:125–130, 1965.

[48] J. Edmonds and E.L. Johnson. Matching, euler tours and the chinese postman. *Mathematical Programming*, 8:88–124, 1973.

[49] H. A. Eiselt, M. Gendreau, and G. Laporte. Arc routing problems, part ii: The rural postman problem. *Operational Research*, 43:399–414, 1995.

[50] L. Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii Academiae Petropolitannae*, 8:128–140, 1736.

[51] D. Feillet, P. Dejax, and M. Gendreau. The profitable arc tour problem: Solution with branch and price algorithm. *Transportation Science*, 39(4):539–552, 2005.

[52] D. Feillet, P. Dejax, and M. Gendreau. Traveling salesman problems with profits. *Transportation Science*, 39(2):188–205, 2005.

[53] E. Fernández, R. Garfinkel, O. Meza, and M. Ortega. On the undirected rural postman problem : Tight bounds based on a new formulation. *Operations Research*, 51:281–291, 2003.

[54] M. Fischetti, J. Salazar, and P. Toth. The generalized traveling salesman and orienteering problems. In G. Gutin and A. Punnen, editors, *The Traveling Salesman Problem and Its Variations*. Kluwer Academic Publishers, 2002.

[55] B. Fleischmann. A cutting plane procedure for the traveling salesman problem on a road network. *European Journal of Operational Research*, 21:307–317, 1985.

[56] G. N. Frederickson, M. S. Hecht, and C. E. Kim. Approximation algorithms for some routing problems. *SIAM Journal on Computing*, 7(2):178–193, 1978.

[57] G.N. Frederickson. Approximation algorithms for some postman problems. *Journal of ACM*, 7:178–193, 1979.

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

[59] B.L. Golden and R.T. Wong. Capacitated arc routing problems. *Networks*, 11:305–315, 1981.

[60] R.E. Gomory and T.C. Hu. Multiterminal network flows. *SIAM journal of Applied Mathematics*, 9:551–556, 1961.

[61] P. Greistorfer. Solving mixed and capacitated problems of the chinese postman. *Central European Journal for Operations Research and Economics*, 3(4):285–309, 1995.

[62] M. Grötschel and O. Holland. A cutting plane algorithm for minimum perfect 2- matchings. *Computing*, 39:327Ű344, 1987.

[63] M. Grötschel and M.W. Padberg. *Polyhedral Theory. The Traveling Salesman Problem: A guided Tour of Combinatorial Optimization.* Wiley, Chichester, 1985.

[64] M. Grötschel and Z. Win. A cutting plane algorithm for the windy postman problem. *Mathematical Programming*, 55:339–358, 1992.

[65] M. Guan. Graphic programming using odd and even points. *Chinese Mathematics*, 1:273–277, 1962.

[66] M. Guan. On the windy postman problem. *Discrete Applied Mathematics*, 9:41–46, 1984.

[67] G. Gutin and A.P. Punnen (eds). *The Traveling Salesman Problem and its variations.* Kluwer Academic Publishers, 2002.

[68] J.H. Hamilton. Memorandum respecting a new system of roots of unity: the icosian calculus. *Philosophical Magazine*, 12:446, 1856.

[69] A. G. Hertz, P. Laporte, and H. Nanchen. Improvement procedures for the undirected rural postman problem. *INFORMS J.Comput.*, 1:53–62, 1999.

[70] C. Hierholzer. On the possibility of traversing a line system without repetition or discontinuity. *Mathematische Annalen*, 6:30–32, 1873.

[71] R. Hirabayashi, Y. Saruwatari, , and N. Nishida. Tour construction algorithm for the capacitated arc routing problems. *Asia Pacific Journal of Operational Research*, 9(2):155–175, 1992.

[72] M. Jünger, G. Reinelt, , and G. Rinaldi. Handbook in operations research and management science: Net- work models. In M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, editors, *The traveling salesman problem*. Elsevier, 1995.

[73] L.R. Ford Jr. and D.R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[74] R.M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972.

[75] T.P. Kirkman. On the representation of polyhedra. *Philosophical Transactions of the Royal Society of London Ser. A*, 146:413–418, 1856.

[76] G. Laporte. Modelling and solving several classes of arc routing problems as traveling salesman problems. *Computers & Operations Research*, 24(11):1057–1061, 1997.

[77] G. Laporte, H. Mercure, and Y. Norbert. Finding the hamiltonian circuit through n clusters. *Congr. Numer.*, 48:227–290, 1985.

[78] G. Laporte, H. Mercure, and Y. Norbert. Generalized traveling salesman problem through n sets of nodes: The asymmetrical case. *Discrete Applications Mathematics*, 18:185–197, 1987.

[79] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. *The Traveling Salesman Problem*. Wiley-Interscience, 1985.

[80] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Khan, and D.B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley, 1985.

[81] J.K. Lenstra and A.H.G. Rinnooy Kan. On general routing problem. *Networks*, 6:273–280, 1976.

[82] A. N. Letchford. New inequalities for the general routing problem. *European Journal of Operational Research*, 96:317–322, 1997.

[83] A. N. Letchford. The general routing polyhedron: A unifying framework. *European Journal of Operational Research*, 112:122–133, 1999.

[84] A. N. Letchford and R. W. Eglese. The rural postman problem with deadline classes. *European Journal of Operational Research*, 105:390–400, 1998.

[85] A.N. Letchford, G. Reinelt, and D.O. Theis. A faster exact separation algorithm for blossom inequalities. *Integer Programming and Combinatorial Optimization 10*, 3064 of LNCS:196–205, 2004.

[86] G. Reinelt M. Jünger and S. Thienel. Practical problem solving with cutting plane algorithms in combinatorial optimization. In L. Lovász W. Cook and P. Seymour, editors, *Combinatorial Optimization*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, page 111Ũ152. Providence, 1995.

[87] E. Minieka. The chinese postman problem for mixed networks. *Management Science*, 25(7):643–648, 1979.

[88] E. Noon and J.C. Bean. A lagrangean based approach to the asymmetric generalized traveling salesman problem. *Operations Research*, 39:623–632, 1991.

[89] E. Noon and J.C. Bean. An efficient transformation of the generalized traveling salesman problem. *INFOR*, 31:39–44, 1993.

[90] Y. Norbert and J. Picard. An optimal algorithm for the mixed chinese postman problem. *Networks*, 27(2):95–108, 1996.

[91] GNU open software. Gnu linear programming kit (glpk). reference manual. *Version 4.8*, 2005.

[92] C. C. Orloff. A fundamental problem in vehicle routing. *Networks*, 4:35–64, 1974.

[93] M. Padberg and M.R. Rao. Odd minimum cut-sets and b-matchings. *Mathematics of Operations Research*, 1982.

[94] M. Padberg and G. Rinaldi. Facet identification for the symmetric traveling salesman polytope. *Mathematical Programming*, 47:219–257, 1990.

[95] C. H. Papadimitriou. On the complexity of edge traversing. *Journal of the ACM*, 23:544–554, 1976.

[96] W. L. Pearn. Solvable cases of the k-person chinese postman problem. *Operations Research Letters*, 16(4):241–244, 1994.

[97] W. L. Pearn, A. Assad, and B. L. Golden. Transforming arc routing into node routing problems. *Computers & Operations Research*, 14(4):285–288, 1987.

[98] W. L. Pearn and C. M. Liu. Algorithms for the chinese postman problem on mixed networks. *Computers & Operations Research*, 22(5):479–489, 1995.

[99] I. Plana. El problema general de rutas con viento. PhD thesis, Universat de València, Spain, 2005.

[100] B. Raghavachari and J. Veerasamy. Approximation algorithms for the mixed chinese postman problem. In E. A. Boyd R. E. Bixby and R. Z. Rios-Mercado, editors, *Integer Programming and Combinatorial Optimization, 6th International IPCO Conference*, volume 1412 of *Lecture Notes in Computer Science*, pages 169–179. Springer, Houston, Texas, 1998.

[101] A. Romero. On mixed rural postman problem (in spanish). PhD thesis, University of Valencia, 1997.

[102] M. Sanchis. El poliedro del problema del cartero rural. PhD thesis, Universidad de Valencia, Spain, 1990.

[103] A. Schrijver. *Min-Max results in Combinatorial Optimization. Mathematical Programming- The State of the Art.* Springer, Heidelberg, 1983.

[104] P. Toth and D. Vigo. *The Vehicle Routing Problem.* SIAM Monographs on Discrete Mathematics and Applications, 2002.

[105] S. A. Welz. Optimal solutions for the capacitated arc routing problem using integer programming. PhD thesis. University of Cincinnati, 1994.

[106] Z. Win. Contributions to routing problems. PhD thesis, University of Augsburg, 1987.

[107] Z. Win. On the windy postman problem in eulerian graphs. *Mathematical Programming*, 44:97–112, 1989.