Solving the Prize-collecting Rural Postman Problem

Julián Aráoz
Simón Bolívar University*, Venezuela
Technical University of Catalonia†, Spain

Elena Fernández
Technical University of Catalonia, Spain

Oscar Meza‡
Simón Bolívar University, Venezuela.

Abstract

In this work we present an algorithm for solving the Prize-collecting Rural Postman Problem. This problem was recently defined and is a generalization of other arc routing problems like, for instance, the Rural Postman Problem. The main difference is that there are no required edges. Instead, there is a profit function on the edges that must be taken into account only the first time that an edge is traversed.

The problem is modeled as a linear integer program where the system has an exponential number of inequalities. We propose a solution algorithm where iteratively we solve relaxed models with a small number of inequalities, that provide upper bounds, and we propose exact separation procedures for generating violated cuts when possible. We also propose a simple heuristic to generate feasible solutions that provide lower bounds at each iteration. We use a two-phase method with different solvers at each phase.

Despite the difficulty of the problem, the numerical results from a series of computational experiments with various types of instances assess the good behavior of the algorithm. In particular, 75% of the instances were optimally solved with the LP relaxation of the model. The remaining instances were optimally solved on a second phase, most of them in small computation times.

*Retired Professor Dpt. Process and Systems.
†Visiting Professor Dpt. Statistics and Operation Research.
‡Retired Professor Dpt. Computation and Information Technology

1 Introduction

In this work we present an algorithm to solve the Prize-collecting Rural Postman Problem (PRPP). This problem was defined in Aráoz, Fernández and Zoltan [5] with the name Privatized Rural Postman Problem, and is part of the family of Arc Routing Problems (ARPs), that usually aim to determine a least-cost traversal of a specified arc subset of a graph, with or without constraints. Such problems arise in a variety of practical contexts, like post delivery and garbage collection. Typical constraints of ARPs require the routes to begin and to finish at a given point (the depot), and guarantee the connectivity of the solutions with the depot. We refer the reader to Dror [17] for a comprehensive state on the art of such problems. Here we consider only Edge Routing Problems (ERPs), which are the undirected cases of ARPs.

Like in other ERPs, in the PRPP we assume that the demand of service is placed at the edges of a graph. However, there is no specific arc subset to be traversed. Instead, we assume that giving service to an edge not only would incur a cost (associated with displacement), but would also result on a profit (associated with servicing edges). The displacement cost to an edge will certainly account for the cost of all the edges that are traversed in the same route, as many times as they are traversed. Similarly, the profit associated with servicing an edge, must also take into account the profit of the additional edges that are serviced in the same route. However, the profit of each edge serviced in the route will be collected only once, independently of the number of times the edge is traversed. In the PRPP we look for traversals that maximize the total servicing profit minus the displacement cost. As shown in [5] the Rural Postman Problem (RPP) is a particular case of PRPP. In fact, PRPPs constitute a generalization of most ERPs with one single route and is equivalent to most Traveling Salesman Problems (TSPs).

The motivation for studying the PRPP comes not only from its theoretical interest but also from its potential applications. In general, the applications of classical ERPs defined as minimization problems arise in public administrations and assume that it has been decided a priori that the service to be given is essential at certain demand sites. Thus, the goal is not to decide the edges to attend, but which is a least cost traversal. However, in the context of private companies looking to maximize operational profits, a demand edge would not be serviced unless it would result on a benefit for the company. As an example we can mention the collection of recycling bins by a private entity. Additionally, problems of this type might appear as subproblems in the pricing stage when addressing capacitated ERPs by column generation.

To the best of our knowledge, the literature on ERPs with profits on the edges is scarce. Apart from [5] and our previous works [3, 2], we are aware of only two works, one by Deitch and Ladany [16] and another one by Feillet, Dejax and Gendreau [19] where problems of this type are considered. In both papers the setting and the approach are quite different from ours: while our approach addresses the problem as an arc-routing one and exploits the fact that only the first traversal of edges is beneficial, in [16] the problem is transformed into a node-routing one, and in [19] a limit, possibly greater than one, is given on the number of times the visit is beneficial on an edge. We are not aware of any work where a solution method is proposed for the PRPP.

In this work we model the PRPP by means of a linear system with an exponential number of inequalities, based on the one defined in [5], and we propose an algorithm to solve the problem. The algorithm is composed of two phases. Each phase consists of an iterative process, where at each iteration the following steps are performed:

i) Solving a relaxation of PRPP;

ii) Applying a simple heuristic (an adaptation of the 3T heuristic of Fernández, Meza, Garfinkel and Ortega [18] for the Rural Postman Problem) to find a feasible solution;

iii) Solving the separation problem for finding different types of inequalities violated by the optimal solution to the current relaxation; and,

iv) Adding the violated inequalities to the current system.

Thus, at the end of each iteration we have an upper bound to the original problem (given by the solution to the relaxed problem), and a lower bound (given by the value of the feasible solution obtained with the heuristic). Since the original system has an exponential number of inequalities, at both phases we consider relaxations of PRPP where only a small subset of the inequalities are included. The difference between the two phases is the type of relaxation that is considered: Linear Programming (LP) problems at the first phase, and integer problems (IP) at the second phase. Both phases terminate either when optimality can be proven or when no more violated cuts can be generated. A flowchart with a general scheme of the algorithm is depicted in Figure 1.

FIGURE 1 HERE

Figure 1: Flowchart of the solution algorithm.

Despite the difficulty of the problem, the numerical results from a series of computational experiments with various types of instances assess the good behavior of the algorithm, since in most cases the problems are solved in small computation times.

The paper is structured as follows. In Section 2 we define the problem and recall from [5] some properties of its solutions. The formulation that we use is presented in Section 3. Section 4 describes the steps of the iterative process, including the initial relaxation of the problem. The exact separation algorithms, both for the inequalities that guarantee the parity of the nodes and for the inequalities that guarantee connectivity of the solutions with the depot are given in Section 5. Section 6 describes the heuristic that has been used to obtain feasible solutions and, thus, lower bounds. In Section 7 we describe the computational experiments and present the obtained results. We finish the paper in Section 8 with the conclusions and some remarks.

2 The Prize-collecting Rural Postman Problem

2.1 Notation and Nomenclature

As usual $G = (V, E)$ is a undirected graph with vertex set V, $|V| = n$, and edge set E, $|E| = m$. For any subgraph H we denote by $V(H)$ the vertex set of H and by $E(H)$ the edge set of H, in particular, $V = V(G),\ E = E(G)$. Let S be a non-empty proper subset of V, we denote by:

$\delta(S) = \{e \in E | e = (u, v), u \in S, v \in V \setminus S\} = \delta(V \setminus S)$, the edges in the cut between S and $V \setminus S$, and

$\gamma(S) = \{e \in E | e = (u, v), u, v \in S\}$, the edges with both vertices in S.

Therefore given $\emptyset \subset S \subset V$, E is partitioned in three sets $\gamma(S),\ \delta(S) = \delta(V \setminus S)$, and $\gamma(V \setminus S)$. For a singleton set we do not use the brackets and we write $\delta(v) \equiv \delta(\{v\})$.

For any edge set $F \subseteq E$, the set of vertices incident to edges in F is denoted $V(F)$. The subgraph induced by F is $G(F) = (V(F), F)$. For any vertex set $U \subseteq V$ we denote $G[U]$ the induced graph $(U, \gamma(U))$.

We use the standard compact notation $f(A) \equiv \sum_{e \in A} f_e$ when $A \subseteq E$, and f is a vector or a function defined on E. If f is defined on a subset $B \subset E$ we also use $f(A) \equiv f(A \cap B) \equiv \sum_{e \in A \cap B} f_e$. Also, by extension, if H is a graph we denote $f(H) \equiv f(E(H))$.

We next recall from [5] the definitions and results relevant for our algorithm.

Definition 2.1 Prize-collecting Rural Postman Problem (PRPP).

Given a graph $G(V, E)$ with a distinguished vertex d, the Depot, and two functions from E in $\mathbb{R}_+$, the profit function b and the cost function c, the Prize-collecting Rural Postman Problem is to find one closed path $\mathcal{C}^*$ which maximizes the value of

$$\sum_{e \in \mathcal{C}} (b_e - t_e c_e)$$

where $\mathcal{C}$ is a closed path in G passing trough d, and not necessarily simple, and t_e is the number of times that edge e is traversed in $\mathcal{C}$.

We denote a PRPP by (G, d, b, c).

That is, in a PRPP, we look for the most Profitable Subtour passing through d. Like in other arc routing problems defined on undirected graphs, in PRPP we allow for consecutive traversals of the same edge in both directions, and trees traversed two times define feasible solutions. In [5] it was proven that, in general, the PRPP is NP–Hard. In [5] it was also proven that when the underlying graph is a tree, then the PRPP can be solved in polynomial time.

We define the functions $\varphi_e = b_e - c_e$ and $\psi_e = b_e - 2c_e = \varphi_e - c_e$. Since, the value φ_e is the net profit obtained the first time that edge e is traversed, and $-c_e$ is the net profit in each of the subsequent traversals of edge e, ψ_e is the the net profit of edge e when it is traversed twice. In the PRPP the edges in $R = \{e \in E \mid \psi_e \geq 0\}$ play an important role, similar to the set of required edges in a RPP.

We denote by $G_R \equiv (V(R) \cup \{d\}, R)$ the subgraph induced by the edge set R and the depot. Let $C_k, k \in P = \{0, \ldots, p\}$ be the connected components of the graph G_R and we assume that $d \in C_0$. Using standard graph nomenclature, when necessary, we denote by $V_k = V(C_k)$, and we differentiate between $\gamma(V_k)$ in the original graph G and $\gamma_R(V_k)$ in the graph G_R. That is $\gamma_R(V_k) = \gamma(V_k) \cap R$. Similarly, for $S \subset V$, $\delta_R(S) = \delta(S) \cap R$.

Let $\mathcal{C}^*$ denote an optimal solution. The following properties hold:

2.2 *Dominance 1. No edge is used more than twice in $\mathcal{C}^*$.*

2.3 *Dominance 2. If an edge $e \in E$ is used with value c_e in $\mathcal{C}^*$, then it is also used with value φ_e in $\mathcal{C}^*$.*

2.4 *Dominance 3. Let $e \in \mathcal{C}^*$, if for some connected component C_k of G_R we have that $V(e) \cap V_k \neq \emptyset$ then all the edges of $\gamma_R(V_k)$ are in $\mathcal{C}^*$.*

Remark 2.5 *Dominance 3 implies that either all the edges of $\gamma_R(V_k)$ are in $\mathcal{C}^*$, or none of them is in $\mathcal{C}^*$. Throughout, for each $k \in P$, e_k^R is chosen to be one of the edges in C_k with greatest φ_e.*

Note that Dominance 3 also implies that if any edge not in R incident with a vertex in V_k is used, then all the edges in $\gamma_R(V_k)$ will be used.

2.6 *Preprocessing 1. Let C_k be one of the connected components of G_R. If $e \in \gamma(V_k) \setminus R$ then e is used at most once in $\mathcal{C}^*$.*

Remark 2.7 *Dominance 3 together with Preprocessing 1, imply that if an edge in $e \in \gamma(V_k) \setminus R$ is in $\mathcal{C}^*$, then also all the edges $\gamma_R(V_k)$ are in $\mathcal{C}^*$.*

From a more general result in [5] we also have the following corollary:

2.8 *Preprocessing 2. $\gamma_{\mathbf{R}}(\mathbf{V_0}) \subseteq \mathcal{C}^*$.*

3 Formulation

The linear equations that we use to define the feasible solutions are based on the ones given in the paper of Aráoz, Férnandez and Zoltan [5].

The model is defined on the graph G. Each edge $e \in E$ is associated with variable x_e corresponding to the first time edge e is used, and with variable y_e corresponding to the second time the edge e is used.

The Linear Integer Program for PRPP is the following:

$$
\begin{aligned}
(M0) \quad & \max z(x,y) = \sum_{e\in E} \varphi_e x_e - \sum_{e\in E} c_e y_e && (1)\\
& x(\delta(S)\setminus F) + y(F\setminus L) \geq x(F) + y(L) - (|F|+|L|) + 1, \\
& \qquad S \subset V\setminus\{d\},\ F\subseteq\delta(S),\ L\subseteq F,\ |F|+|L| \text{ odd.} && (2)\\
& x(\delta(S)) + y(\delta(S)) \geq 2x_e, \\
& \qquad S\subseteq V\setminus\{d\},\ e\in\gamma(S),\ V(e)\cap V(R)=\emptyset && (3)\\
& x(\delta(S)) + y(\delta(S)) \geq 2x_{e_k^R}, \\
& \qquad S\subseteq V\setminus\{d\},\ k\neq 0,\ V_k\cap S\neq\emptyset && (4)\\
& x_e = x_{e_k^R}, \qquad e\in\gamma(V_k)\cap R,\ k\in P && (5)\\
& y_e \leq x_e, \qquad e\in E && (6)\\
& x_e \leq x_{e_k^R}, \qquad e\in((\gamma(V_k)\cup\delta(V_k))\setminus R),\ k\in P && (7)\\
& y_e = 0, \qquad e\in\gamma(V_k)\setminus R,\ k\in P && (8)\\
& x_e = 1, \qquad e\in\gamma(V_0)\cap R && (9)\\
& x_e, y_e \in \{0,1\}, \qquad e\in E && (10)
\end{aligned}
$$

- Inequalities (2) ensure even degree at every subset of vertices and, in particular, at every vertex. We call set-parity inequalities to constraints (2), and node-parity inequalities to the particular case with S being a singleton, i.e. $S=\{v\}$, $v\in V$. Inequalities (2) are an adaptation to our problem of the so called co-circuit inequalities introduced by Barahona and Grötschel [6] in a matroid context (see also Aráoz et al. [1]). 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 of the subset of vertices $\delta(S)$. In our case, we further exploit the precedence relationship of x variables with respect to y variables given by Dominance 2, 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) can be rewritten as

$$
\begin{aligned}
x(\delta(S) \setminus F) + y(F \setminus L) \geq x_e + 1 - 1 \quad &\equiv \\
x(\delta(S)) - x_e + y_e \geq x_e \quad &\equiv \\
x(\delta(S)) + y_e \geq 2x_e \quad &(11)
\end{aligned}
$$

We denote this particular case as parity-connectivity inequalities.

- Inequalities (3) and (4) ensure connectivity with the depot of any edge $e \in \gamma(S)$ that is selected (that is, such that $x_e = 1$). By Dominance 2 we do not need to check connectivity when $y_e = 1$, and by Dominance 3 for the edges in C_k we only need to check for e_k^R. These inequalities are referred to as connectivity inequalities. Inequalities (3) and (4), together with inequalities (2), guarantee that the solution to the model define Eulerian circuits.
- Equalities (5) correspond to Dominance 3.
- Inequalities (6) correspond to Dominance 2.
- Inequalities (7) are also implied by Dominance 3 (see Remark 2.7).
- Equalities (8) correspond to Preprocessing 1.
- Equalities (9) correspond to Preprocessing 2.
- Binary conditions (10) correspond to Dominance 1.

4 Algorithm

In this section we describe the algorithm that we apply to solve PRPP. Different types of routing problems are frequently modeled using families of constraints that have an exponential number of inequalities. In these cases, the models are too big to be given to any solver, even for obtaining LP bounds when integrality constraints are relaxed. Several approaches have been proposed and used for this type of systems, among which Branch and Cut is probably the most common one.

In the case of the proposed model for PRPP, the families of constraints (2), (3), (4) have an exponential number of inequalities. Hence, our goal is to find a relatively small subset of these inequalities that determines the optimal vertex. The algorithm that we propose has two phases. Both phases are based on an iterative scheme like the one depicted in Figure 1. The difference between the two phases is the type of relaxation to PRPP that is considered at each iteration. Both relaxations are *constraint relaxations* of PRPP in the sense that only a subset of constraints (2), (3), (4) are considered. In the first phase we additionally relax integrality constraints on the variables, so that the problem that is solved at each iteration is an LP problem, whereas in the second phase, integrality conditions on the variables are kept, so that the relaxation of PRPP is an Integer Problem (IP) with binary variables. For ease of presentation in Figure 1, within the same phase, we denote RM0 to the input and RMr to the problem that is solved at iteration r of the corresponding phase. The input of the first phase will be described in the next

subsection. The input of the second phase is the problem that is solved in the last iteration of the first phase strengthened with the integrality conditions of the variables.

In both phases we solve exactly the separation problem to generate violated cuts associated with constraints (2), (3), (4), when they exist. Each of the two phases terminates when at a given iteration there are no more cuts violated by the solution to the current relaxation or if optimality of the best solution found so far is proven. When the first phase terminates the algorithm enters the second phase, and the algorithm ends when the second phase terminates. Let $\bar{z}$ denote the upper bound, and let *bestxy* be the best solution found so far, with value $\underline{z}$, at the end of an iteration. Optimality is proven when $\bar{z} - \underline{z} = 0$. In this case, *bestxy* is the optimal solution. At the end of the first phase, when optimality has not been proven, integrality conditions are posed on the variables of the current LP, and the algorithm enters the second phase. A scheme of a phase of the algorithm is given in Figure 2. We next describe its main components.

CutProcedure($r = 0$; $\bar{z}$; $\underline{z}$; *bestxy*)

1. *endphase*= *false*;
2. **while** *endphase*= *false* **do:**
3. **begin**
   - (a) Find a solution (x^*, y^*) of RMr; $z(x^*, y^*)$ denotes its associated cost;
   - (b) **if** $\bar{z} > z(x^*, y^*)$ **then** $\bar{z} = z(x^*, y^*)$;
   - (c) **if**(x^*, y^*) is feasible to PRPP **then**
     bestxy= (x^*, y^*); $\underline{z} = \bar{z}$; *endphase*= *true*;
     else
     - i. Identify inequalities of types (2), (3), (4) violated by (x^*, y^*);
     - ii. Add the valid inequalities identified in step (3.(c)i.) to RMr;
     - iii. Compute the 3T heuristic solution (x^h, y^h); $z(x^h, y^h)$ denotes its associated cost (only in the first phase);
     - iv. **if** $\underline{z} < z(x^h, y^h)$ **then** $\underline{z} = z(x^h, y^h)$; *bestxy* $= (x^h, y^h)$;
     - v. **if** $(\bar{z} = \underline{z})$ **or** (no inequality has been added in step (3.(c)ii.)) **then** *endphase*=*true*;
     - vi. $r := r + 1$
4. **end** (3)
5. **output**(RMr; $\bar{z}$; $\underline{z}$; *bestxy*)

Figure 2: Scheme of each phase of the Algorithm

Observe that the two phase algorithm that we propose can be seen as a branch-and-cut algorithm in which (in phase 2) we resort to a general purpose solver for deciding the strategies on the branching variables and the selection of subproblems. In this algorithm we only generate cuts associated with inequalities (2), (3) and (4) at the root node (phase 1) or when some integer solution is obtained. At the rest of the nodes of the search tree (phase 2) the general purpose solver possibly generates Gomory cuts that are not included in our model.

4.1 Initial LP Relaxation

The cutting plane algorithm starts with the following LP:

$$
\begin{aligned}
&(LPM0)\\
&\max z(x,y) = \sum_{e\in E} \varphi_e x_e - \sum_{e\in H} c_e y_e\\
&x(\delta(v)) + y_e \geq 2x_e, \qquad v \in V, e \in \delta(v).\\
&x(\delta(V_k)) + y(\delta(V_k)) \geq 2x_{e_k^R}, \qquad k \in P \setminus \{0\}\\
&x_e = x_{e_k^R}, \qquad e \in \gamma_R(V_k) \text{ and } k \in P \setminus \{0\}\\
&y_e \leq x_e, \qquad e \in E\\
&x_e \leq x_{e_k^R}, \qquad e \in ((\gamma(V_k) \cup \delta(V_k)) \setminus R),\ k \in P\\
&x_e = 1, \qquad e \in \gamma(V_0) \cap R\\
&0 \leq x_e \leq 1, \qquad 0 \leq y_e \leq 1, \qquad e \in E
\end{aligned}
$$

This initial program, in which integrality conditions (10) are relaxed, has the same functional (1) and all the inequalities (5)-(9) of M0. LPM0 only includes a subset of inequalities of type (2), namely $\delta(v) + y_e \geq 2x_e$, $v \in V, e \in \delta(v)$, which are the parity-connectivity inequalities associated with singletons $S = \{v\}$, $v \in V$. LPM0 does not include any inequality (3), but it includes the subset of the connectivity inequalities (4) associated with the sets $S = V_k$, $e = e_k^R$.

It is important to note that, since some of the original constraints are omitted, the solutions to this system need not be feasible to the PRPP even if they were integer.

5 Separation of Cuts

At a given iteration of the algorithm the step that solves exactly the separation problem for the relaxed inequalities consists of two parts: first, we look for violated inequalities of type (2), and second, we look for violated inequalities of type (3) and (4). Throughout $G_{x^*,y^*} = (V, E_{x^*,y^*})$ denotes the support graph of the current solution (x^*, y^*), obtained from the graph G by eliminating all edges in E with $x_e^* = 0$. Our separation algorithms for inequalities (2), will be presented first for general co-circuit inequalities with generic variables x. Then $G_{x^*} = (V, E_{x^*})$ will denote the support graph with respect to a given vector x^*.

5.1 Finding violated inequalities of type (2)

First, we give exact separation algorithms for two subsets of inequalities (2), namely *i*) node-parity inequalities, that is $S = \{v\}$, $v \in V$ and general subsets $F \subseteq \delta(v)$, $|F|$ odd, and, *ii*) parity-connectivity inequalities. Then, we give the exact separation for the general case.

5.1.1 Node-parity inequalities

For separating node-parity inequalities we use a modification of a heuristic proposed by Ghiani and Laporte [22] for the RPP. This modification results in an exact separation method for node-parity inequalities for the PRPP. First, we give the separation algorithm for co-circuit inequalities when S is a singleton in its general form, and then we particularize it for our node-parity inequalities. Given a graph G and a vector $x : E \to [0, 1]$, in general, the co-circuit inequalities for a singleton $\{v\}$ are:

$$x(\delta(v) \setminus F) \geq x(F) - \mid F \mid +1, \ \forall F \subseteq \delta(v), \ |F| \text{ odd} \tag{12}$$

The separation problem for these inequalities is: Given a vector x^*, $0 \leq x^* \leq 1$ to determine if there exists some inequality of type (12) violated by x^*, or to determine that none exists.

5.1 Algorithm:

1. For each vertex v do
   (a) Let $F = \{e \in \delta(v) | x^*_e \geq 0.5\}$.
   (b) If F is even then let $x^*_{e^1} = \min\{x^*_e | e \in F\}$ and $x^*_{e^2} = \max\{x^*_e | e \in \delta(v) \setminus F\}$. If $x^*_{e^1} - 0.5 \leq 0.5 - x^*_{e^2}$ delete e^1 from F otherwise add e^2 to F.
   (c) If the inequality (12) corresponding this set F is violated by x^*, then we have a cut.

Theorem 5.2 *(Aráoz, Fernández and Meza [4]) The separation Algorithm (5.1) is exact and the order of the algorithm is $|E|$.*

Remark 5.3 In our problem we have variables x and y. Hence, step (1a) of Algorithm 5.1 must be adapted by defining $F = \{e \in \delta(v) | x^*_e \geq 0.5\}$ and $L = \{e \in \delta(v) | y^*_e \geq 0.5\}$. Thus, in step (1a) we have $L \subseteq F$ since, by inequalities (6), $y^*_e \geq 0.5 \Rightarrow x^*_e \geq 0.5$. In step (1b), we can keep $L \subseteq F$ as follows:

If $|F|+|L|$ is even, then consider $z_{e^1} = \min\{\{x^*_e : e \in F\}, \{y^*_e : e \in L\}\}$ and $z_{e^2} = \max\{\{x^*_e : e \in \delta(v) \setminus F\}, \{y^*_e : e \in \delta(v) \setminus L\}\}$. Suppose $z^*_{e^1} - 0.5 \leq 0.5 - z^*_{e^2}$. If $e^1 \in L$, then $L := L \setminus \{e^1\}$, and $L \subseteq F$; otherwise, updating $F := F \setminus \{e^1\}$ we still have $L \subseteq F$. Suppose now that $z^*_{e^1} - 0.5 > 0.5 - z^*_{e^2}$. Similarly, to the above case if $e^2 \notin F$, then $F := F \cup \{e^2\}$; otherwise, $L := L \cup \{e^2\}$, and in both cases $L \subseteq F$.

If

$$x^*(\delta(v) \setminus F) + y^*(F \setminus L) + \sum_{e \in F}(1 - x^*_e) + \sum_{e \in L}(1 - y^*_e) - 1 < 0$$

then the following inequality is violated by (x^*, y^*):

$$x(\delta(v) \setminus F) + y(F \setminus L) \geq x(F) + y(L) - |F| - |L| + 1$$

5.1.2 Parity-connectivity inequalities

For separating parity-connectivity inequalities we first obtain the tree of min-cuts (see, for instance, [23]) of the support graph G_{x^*,y^*}, after assigning to each edge e a capacity of value $(x_e^* + y_e^*)$. Then, the parity-connectivity inequalities are separated exactly with a very simple algorithm: For each edge $e = \{u, v\} \in E$ we determine the cut $\delta(S)$ in G_{x^*,y^*}, that separates v and u, which is minimum relative to the assigned capacities. If the value of this min-cut is less than $2x_e$ then the following inequality is violated by (x^*, y^*): $x(\delta(S)) + y(\delta(S)) \geq 2x_e$. The order of this algorithm is dominated by that of the algorithm to obtain the cut tree, which is $\mathcal{O}(|V|^4)$.

5.1.3 General case

Set-parity inequalities (2) can be separated exactly for general sets S and $F \subset S$, $|F|$ odd in polynomial time. In [24] Grötschel and Holland [24] propose an algorithm for separating Set-parity inequalities, which was applied in the context of Perfect 2-Matchings. Belenguer and Benavent [8] and Benavent et al. [12] have used this exact algorithm in the context of arc-routing problems. Other authors have considered heuristics to separate some particular cases. For instance, Ghiani and Laporte [22] proposed a heuristic for finding node-parity inequalities in the context of the RPP. The separation algorithm of [24] requires to apply the Padberg-Rao algorithm [26] to an *ad hoc* graph, that contains more nodes and edges than the support graph of the given solution. In particular, if $G_{x^*} = (V, E_{x^*})$ denotes the support graph of a given vector x^*, $0 \leq x_e^* \leq 1$ $e \in E$, the *ad hoc* graph used in the algorithm of [24] has $|V| + K$ vertices and $|E_{x^*}| + K$ edges, where K denotes the number of non integer components of x^*. Thus, the order of the algorithm is $(O\left((|V| + |E|)^4\right))$ (see [24]). The exact separation algorithm that we use in this work has been proposed recently by Aráoz, Fernández and Meza [4]. This separation algorithm requires to find the min-cut tree of the original support graph of G_{x^*} relative to *ad hoc* capacities on the edges, and is of order $(O(|V|)^4)$. Next we present the separation algorithm of Aráoz, Fernández and Meza [4] for co-circuit inequalities in their general form and then we particularize for the case of our inequalities (2) with x and y variables. Given a graph G and a vector $x : E \rightarrow [0, 1]$, the general form of inequalities (2) for a subset of vertices $S \subset V$ is:

$$x(\delta(S) \setminus F) \geq x(F) - \mid F \mid +1, \ \forall F \subseteq \delta(S), \ |F| \text{ odd},$$

that can be rewritten as

$$\sum_{e \in \delta(S) \setminus F} x_e + \sum_{e \in F} (1 - x_e) \geq 1, \ \forall F \subseteq \delta(S), \ |F| \text{ odd}, \tag{13}$$

For a given vector x^*, the separation problem for inequalities (13) is to find $S^* \subset V$, $F^* \subseteq \ \delta(S^*)$, $|F^*|$ odd such that the associated inequality (13) is violated by x^*, or to prove that no such inequality exists.

For separating inequalities (13), each edge in E_{x^*} is assigned a capacity given by

$$h_e = \min\{x_e^*, \ 1 - x_e^*\}.$$

We partition E_{x^*}, $E_{x^*} = E^< \cup E^>$ so that an edge e belongs to $E^<$ (resp. $E^>$) if and only if its capacity is given by x^*_e (resp. $1 - x^*_e$). Values 0.5 are assigned arbitrarily to $E^<$ or to $E^>$. We say that a vertex $v \in V$ is *odd*$^>$ if and only if $|\delta_{E^>}(v)|$ is odd; otherwise we say that v is *even*$^>$. By extension, we say that a subset $S \subseteq V$ is *odd*$^>$ if it contains an odd number of vertices labelled odd, i.e. $|\{v \in S : v \text{ is odd}\}|$ is odd.

Lemma 5.4 *(Aráoz, Fernández and Meza [4])*

(a) For $S \subset V$, $|\delta_{E^>}(S)|$ is odd if and only if S is odd$^>$.

(b) V is even$^>$.

Lemma 5.5 *(Aráoz, Fernández and Meza [4]) Let S be a vertex subset, the following two proprieties hold:*

(a) When S is odd$^>$ then a subset F, $|F|$ odd, that minimizes the value of the left hand side of constraint (13) is given by $F = \delta_{E^>}(S)$. The minimum value of the left hand side of the inequality 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 (13) 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 $x^*_{e^1} = \min\{x^*_e : e \in \delta_{E^>}(S)\}$. Otherwise, $F = \delta_{E^>}(S) \cup \{e^2\}$, with e^2 such that $x^*_{e^2} = \max\{x^*_e : e \in \delta(S) \setminus \delta_{E^>}(S)\}$. The minimum value of the left hand side of the inequality 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 $x^*_{e^1} = \min\{x^*_e : e \in \delta_{E^>}(S)\}$ and $x^*_{e^2} = \max\{x^*_e : e \in \delta(S) \setminus \delta_{E^>}(S)\}$. Lemma 5.5 indicates that for a given vector x^*, the smallest value of the left hand side in constraint (13) may correspond *i)* to an *odd*$^>$ set S with $F = \delta_{E^>}(S)$, or *ii)* to an *even*$^>$ set S, with a modified set F defined as indicated in Lemma 5.5(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 $\widehat{h}(\delta(S)) = h(\delta(S)) + \min\{x^*_{e_1} - (1 - x^*_{e_1}), (1 - x^*_{e^2}) - x^*_{e_2}\}$. The following well-known result of Padberg and Rao [26] indicates how to identify the *odd*$^>$ set S with minimum capacity $h(\delta(S))$. First, we give some additional notation.

Suppose that each vertex $v \in V$ is assigned some label λ with two possible values (odd or even). By extension, subsets of vertices also are assigned labels. In particular, for $U \in V$, $\lambda(U)$ is odd iff U contains an odd number of vertices labelled odd. For $S \subset V$, $(S : V \setminus S)$ is an alternative way of denoting the cut-set $\delta(S)$.

Lemma 5.6 *(Padberg and Rao [26]) Let $G = (V, E)$ and $c_e \geq 0$ for all $e \in E$. Let $V = V_0 \cup V_1$ where V_1 is a nonempty set of nodes of G with odd label, V_0 is the set of nodes with even label and $\lambda(V)$ is even. Let $(M : V \setminus M)$ be a minimum cut-set with respect to all pairs of odd labelled nodes in G. Then, there exists an odd minimum cut-set $(X : V \setminus X)$ in G such that $X \subseteq M$ or $X \subseteq V \setminus M$ holds.*

Let $c_e = h_e$, for all $e \in E$, and $V_1 = \{v \in V : v \text{ is } odd^{>}\}$. We assume that $V_1 \neq \emptyset$, since otherwise no $odd^{>}$ set exists. Since V is $even^{>}$ we obtain the following result.

Corollary 5.7 *(Aráoz, Fernández and Meza [4]) Let $G_{x^*} = (V, E_{x^*})$ and $h_e \geq 0$ for all $e \in E_{x^*}$. Let $\delta(M)$ be a minimum cut-set with respect to all pairs of $odd^{>}$ nodes in G_{x^*}. Then there exists an $odd^{>}$ set S with minimum capacity $h(\delta(S))$ such that $S \subseteq M$ or $S \subseteq V \setminus M$ holds.*

As it is well-known, Lemma 5.6 implies that a minimum odd cut-set will be one of the cuts that define the tree of min-cuts of all pairs of odd labelled vertices. Thus, a minimum odd cut-set will also be one of the cuts that define the tree of min-cuts of all pairs of nodes. Hence, Corollary 5.7 implies that an $odd^{>}$ set S with a minimum capacity $h(\delta(S))$ will be one of the cuts that define T, the tree of min-cuts of all pairs of vertices of G_{x^*} relative to the capacities vector h.

We next turn our attention to how to identify an $even^{>}$ set S that minimizes $\widehat{(h)}(\delta(S)) = h(\delta(S)) + \min\{x^*_{e_1} - (1 - x^*_{e_1}), (1 - x^*_{e^2}) - x^*_{e_2}\}$.

Lemma 5.8 *(Aráoz, Fernández and Meza [4]) Let $\delta(\widehat{S})$ be a cut-set with $\widehat{S}$ $even^{>}$.*

a) Let $\hat{e} \in \delta(\widehat{S}) \cap E_{>}$ be such that $h\left(\delta(\widehat{S})\right) - (1 - x_{\hat{e}}) + x_{\hat{e}} < 1$. Then, there exist a cut-set $\delta(S)$ associated with one edge of T, $\hat{e} \in \delta(S)$, and $F \subset \delta(S)$, $|F|$ odd, such that the corresponding inequality (13) is violated by x^*.

b) Let $\hat{e} \in \delta(\widehat{S}) \cap E_{<}$ be such that $h\left(\delta(\widehat{S})\right) - x_{\hat{e}} + (1 - x_{\hat{e}}) < 1$. Then, there exist a cut-set $\delta(S)$ associated with one edge of T, $\hat{e} \in \delta(S)$, and $F \subset \delta(S)$, $|F|$ odd, such that the corresponding inequality (13) is violated by x^*.

The above lemma implies that if there exist an $even^{>}$ set S, and a subset $F \subseteq \delta(S)$, $|F|$ odd, such that its associated inequality (13) is violated by x^*, then, there exists a cut-set in T (the tree of min-cuts of all pairs of vertices of G_{x^*} relative to the capacities vector h) such that its associated inequality (13) is violated by x^*. The result of Lemma 5.8 plays an important role. Due to this result, if there exist violated inequalities (13), one such inequality will be associated to a cut-set of T. Hence, Lemma 5.8 together with Corollary 5.7 guarantee that the separation problem can be solved with order no greater than the procedure to find the min-cuts tree T.

The algorithm that we propose starts building T, the tree of min-cuts of all pairs of vertices of G_{x^*} relative to the capacities vector h. Then the edges in T are explored in turn until a min-cut is found for which the associated inequality (13) is violated, or until all the edges in T have been considered. Each considered min-cut $\delta(S^i)$ is associated with a vertex set S^i, which is is either $odd^{>}$ or $even^{>}$. In the first case, we only have to check if its capacity is smaller than one. If so, $F = \delta_{E^{>}}(S^i)$. When S^i is $even^{>}$, the associated set F will be $F = \delta_{E^{>}}(S^i) \setminus \{e_1\}$ when $x^*_{e_1} - (1 - x^*_{e_1}) < (1 - x^*_{e^2}) - x^*_{e_2}$, or $F = \delta_{E^{>}}(S^i) \cup \{e_2\}$, otherwise. Once F is defined, we have to check if F together with S^i it defines a violated inequality (13). The algorithm is as follows:

5.9 Exact separation algorithm for inequalities (13)

1. Obtain T, the tree of min-cuts of all pairs of odd$^{>}$ vertices of G_{x^*}.
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** $\big(\big(S^i$ is $odd^{>}\big)$ **and** $\big(h(\delta(S^i)) < 1\big)\big)$ **then**
   Terminate. (the inequality (13) with $S = S^i$ and F is violated by x^*).

   3.2 **if** $\big(S^i$ is $even^{>}\big)$ **and** $\big(\widehat{h}(\delta(S^i)) < 1\big)$ **then**
   if $\big(x^*_{e_1} - (1 - x^*_{e_1}) < (1 - x^*_{e^2}) - x^*_{e_2}\big)$ **then**
   $F = \delta_{E^>}(S^i) \setminus \{e_1\}$
   otherwise
   $F = \delta_{E^>}(S^i) \cup \{e_2\}$
   endif
   Terminate. (the inequality (13) with $S = S^i$ and F is violated by x^*).
   endif

   endfor

Theorem 5.10 *(Aráoz, Fernández and Meza [4]) The separation Algorithm 5.9 is exact and the order of the algorithm is* $\mathcal{O}(|V|^4)$.

The order of Algorithm 5.9 is dominated by the computations to obtain the min-cut tree T. T can be obtained with the algorithm of Gomory and Hu, which has an overall order of $|V|^4$ (see [23]). □

Remark 5.11 The adaptation of algorithm 5.9 to inequalities (2) with the x and y variables is similar to that of algorithm 5.1. The capacity that is assigned to each edge $e \in E_{x^*y^*}$ is now given by $h_e = min\{x^*_e, 1-x^*_e\} + min\{y^*_e, 1-y^*_e\}$. Now $E^> = \{e : x^*_e > 0.5\}$, $E^< = \{e : x^*_e < 0.5\}$, and edges with $x^*_e = 0.5$ are arbitrarily assigned to one of the above two sets. For a given node set S, let $F = \delta_{E^>}(S)$, and $L = F \cap \{e : y^*_e > 0.5\}$. Then, S is $odd^{>}$ iff $|F| + |L|$ is odd.

a) When S is $odd^{>}$, $h(\delta(S))$ gives the value of the left hand side of constraint (2) for S, F, and L as defined above.

b) When S is $even^{>}$, consider $z_{e^1} = \min\{\{x^*_e : e \in F\}, \{y^*_e : e \in L\}\}$ and $z_{e^2} = \max\{\{x^*_e : e \in \delta(S) \setminus F\}, \{y^*_e : e \in \delta(S) \setminus L\}\}$.

   b.1) The minimum value of the left hand side of constraint (2) is given by $\widehat{h}(\delta(S)) = h(\delta(S)) + \min\{z^*_{e^1} - 0.5, 0.5 - z^*_{e^2}\}$.

   b.2) Suppose $z^*_{e^1} - 0.5 \leq 0.5 - z^*_{e^2}$. If $e^1 \in L$, then $L := L \setminus \{e^1\}$; otherwise, $F := F \setminus \{e^1\}$.

   b.3) Suppose now that $z^*_{e^1} - 0.5 > 0.5 - z^*_{e^2}$. If $e^2 \notin F$, then $F := F \cup \{e^2\}$; otherwise, $L := L \cup \{e^2\}$.

c) In all the above cases in a) and b) $L \subseteq F$.

FIGURE 3 HERE

Figure 3: Fractional solution that cannot be separated by inequalities (2), (3), (4).

5.2 Finding violated inequalities of type (3) or (4)

For separating any of the inequalities (3) and (4) again we use the tree of min-cuts of the graph G_{x^*} with the capacities given by $x^*_e + y^*_e$ as defined in 5.1.2. Then, we use the exact algorithm of Belenguer and Benavent [8, 9]. The algorithm does the following:

For each edge $e = \{u, v\} \in E$ with u and v different from the depot and $x^*_e > 0$, the minimum cut $\delta(S)$ such that $e \in \gamma(S)$ is easily obtained from the min-cut tree. If the value of the min cut is smaller than $2x^*_e$ then the following inequality is violated by (x^*, y^*): $x(\delta(S)) + y(\delta(S)) \geq 2x_e$.

5.3 Cut generation strategy

The exact separation procedure for node-parity inequalities has order $\mathcal{O}(|E|)$, whereas the rest of the separation procedures are of order $\mathcal{O}(|V|^4)$ since they require to obtain the min-cut tree relative to different capacity functions. The separation procedures for parity-connectivity inequalities and for connectivity inequalities operate on the min-cut tree associated with capacities $x^*_e + y^*_e$, whereas the separation procedure for the general co-circuit inequalities operates on the min-cut tree associated with capacities $min\{x^*_e, 1 - x^*_e\} + min\{y^*_e, 1 - y^*_e\}$. Since for parity-connectivity inequalities and for connectivity inequalities the same min-cut tree is used, at each iteration of the first phase of Algorithm 2 we always apply the separation procedures for node-parity, parity-connectivity, and connectivity inequalities. On the contrary, we only apply the exact separation procedure for the general case of inequalities (2) when no violated inequality of any of the other types has been found.

5.4 Other cuts

Given that the procedures that we apply to separate inequalities (2), (3) and (4) are exact, when no cuts associated with these constraints are violated by the current solution, then either the solution is optimal or it necessarily has fractional values. Hence, there are indeed additional inequalities, other from the ones in system (2)-(8), needed to characterize the convex hull of feasible solutions to PRPP. Figure 3 gives an example of a fractional solution that satisfies constraints (5)-(8) and does not violate any inequality (2), (3), (4). When the current relaxation gives such a point, and the fractional part of some basic variable is above some threshold value, we resort to Gomory cuts [20] to cutoff such a solution.

6 Lower bounds

Next we describe how we obtain feasible solutions, and thus lower bounds for the problem. We have not designed an *ad hoc* heuristic that exploits the specific characteristics of PRPP. Instead, we transform PRPP into a RPP and then we apply the 3T heuristic of Fernández, Meza, Garfinkel and Ortega [18], which gives good results on RPP instances.

The heuristic that we use at each iteration to obtain a feasible solution is the following:

1. Let (x^*, y^*) be the solution of the current linear programming relaxation.
2. Transform the PRPP into a RPP. The graph of the RPP will be the original graph G of PRPP plus fictitious vertex d' and edge $\{d, d'\}$. Edge $\{d, d'\}$ is defined as required to guarantee that the solution passes through the depot. All other edges of PRPP with value x_e^* greater or equal to ϵ are also defined as required edges for RPP (ϵ is a parameter). The cost of edge $\{d, d'\}$ is zero, and any other edge e inherits its cost c_e from PRPP.
3. Find a feasible solution to RPP with the 3T heuristic of Fernández, Meza, Garfinkel and Ortega [18].
4. The feasible solution to PRPP results from eliminating the parallel edges $\{d, d'\}$ from the feasible solution to RPP obtained in step 3.

In the computational experiments, the 3T heuristic is applied twice at each iteration of the first phase: the first time we use $\epsilon = 0.9$. That is, in step 1 we include in the graph of the RPP all the edges with $x_e^* > 0.9$. The second time that we apply the heuristic we use $\epsilon = 0.8$ and include in the graph of the RPP all the edges such that $b_e - 2c_e > 0$ with $x_e^* > 0.8$.

7 Computational Experiments

In order to evaluate the performance of the proposed algorithm we have run a series of computational experiments. We next describe such experiments and give the obtained numerical results. Programs have been coded in C using CPLEX 7.0 library for the solution of the LP relaxations of the first phase and for the solution of the Integer Problems of the second phase. Default parameters have been used.

All instances were run on a Sun ULTRA 10, model 440, 1x440 MHZ, 1GB DRAM. Since there are no available PRPP benchmark instances, we have generated 118 PRPP instances from well known sets of benchmark RPP instances. The 118 RPP benchmark instances are divided in five groups. The first group contains two problems, ALBAIDAA and ALBAIDAB, obtained from the Albaida, Spain Graph (see Corberán and Sanchis [14]). The second group contains the 24 instances (problems labeled P) of Christofides et al. [10]. The last three groups contain instances from Hertz et al [25]: 36 instances with vertices of degree 4 and disconnected required edge sets (labeled D), 36 grid instances (labeled G), and 20 randomly generated instances (labeled R). Table 1 depicts information on these instances, that have been grouped according to their

characteristics and to their sizes. In particular, column under *#inst.* gives the number of instances in the group, columns under $|V|$ and $|E|$ give, respectively, the number of vertices and edges; column under $|R|$ gives the number of edges in the set R, and the last column under $|P|$ gives the number of connected components in the graph G_R. In all columns when all the instances of the group did not have the same values, the minimum and maximum values in the group are given.

In all cases the depot has been taken as vertex 1, and we have kept the cost function c from the original RPP instance. The profits have been generated as follows (U[a,b] denotes the integer uniform distribution in the interval [a,b]):

- $b_e \in$ U[c_e, $3c_e$], if e is a required edge of the RPP instance.
- $b_e \in$ U[0, c_e], if e is a non-required edge of the RPP instance.

The results are summarized in Table 2, although detailed results for each of the instances can be found in http://www-eio.upc.es/ elena/index.html. Columns 2-12 give results corresponding to the first phase. They have been divided in several blocks: Columns 2-4 refer to the LP relaxation and Columns 5-7 to the heuristic. Column under $\#LP$ gives the number of LP iterations in the first phase, and Columns 9-12 give information on the cuts generated. In particular, the column under gap_{RL}, gives the average percent gap at the end of phase 1 with respect to the optimal solution of the upper bound obtained with the LP relaxation. That is, if z_{opt} denotes the value of the optimal solution to the problem, and z_{RL} denotes the values of the LP relaxation at the end of the first phase, the entries of the column gap_{RL} give the average values of the deviations $100 * \frac{z_{RL} - z_{opt}}{z_{opt}}$, over all the instances in each group. Columns under $\#opt_{RL}$, and t_{RL} give, respectively, the number of instances in the group that have been optimally solved with the LP relaxation; and the average cpu-time in seconds.

Similarly, the next three columns under gap_H, $\#opt_H$, and t_H give the average percent gap with respect to the optimal solution of the lower bound obtained with the heuristic (average value of the deviations $100 * \frac{z_{opt} - z_H}{z_{opt}}$), the number of instances in the group that have been optimally solved with the heuristic, and the average cpu-time in seconds, respectively. The following four columns, under *v-par*, *par-conn*, *S-par* and *conn*, depict the average number of generated cuts of node-parity (12), parity-connectivity (11), co-circuit for the general case (2), and connectivity (3)-(4), respectively. The last three columns in the table give results corresponding to the second phase. The column under $\#IP$ gives the average number of iterations performed in the second phase, which corresponds to the average number of integer problems that had to solved. Column under *#nod* shows the average number of nodes that were explored in the search tree of each of the integer problems that had to be solved, and, finally, column t_{IP} gives the total cpu-time in seconds required by the second phase of the algorithm.

In our opinion, the obtained results are very good. As can be seen the optimal solution to the LP relaxation gave the optimal value of the problem for 94 out of the 118 instances. In 87 of these instances optimality could be proven since the solution that gave the bound was already integer. In general, the percent deviation of the LP relaxation from the optimal solution at the end of the first phase is very small, with fifteen instances (in addition to the 94 ones above mentioned) with a percent gap smaller than 1%. There are four instances with a percent gap

between 1% and 2%; three instances with a percent gap between 2% and 5%; one more instance for which this percent gap is between 5% and 10% (in fact, slightly smaller than 7%); and, only one instance (in group PG64) for which this percent gap was over 10% (quite surprisingly, this instance was optimally solved by the heuristic).

Despite its simplicity, the behavior of the heuristic is also very good, since it gave the optimal solution in 92 out of the 118 instances. In most of the cases, the percent deviations from the optimal solution were larger in magnitude than those of the upper bounds, but again there is only one instance (in PP) where the percent gap between the heuristic and optimal solutions is above 10%.

Usually, the number of LP iterations in the first phase of the algorithm is quite small. For 86 out of the 118 instances this number does not exceed 20, and only 10 instances required more than 100 iterations to terminate the first phase. As could be expected, the instances that required more iterations are the ones with a larger number of connected components in the graph induced by the edges in the set R. To some extent the difficulty for solving the instances is also reflected on the number of cuts that were added: both in the total number of cuts and on the number of cuts per iteration. In our opinion the number of generated cuts is small taking into account the difficulty of the problems. In general, the parity-connectivity cuts (11), followed by connectivity cuts (3)-(4), are more frequent than general co-circuit cuts (2) and node-parity cuts (12). Taking into account our strategy for generating cuts, according to which the general co-circuit inequalities are only generated when none of the other cuts were obtained, these figures illustrate the important role of the general co-circuit inequalities in the performance of our algorithm. In fact, a previous version of the algorithm where the exact algorithm for the separation of the general case of co-circuit inequalities was not used, and the only co-circuit inequalities that were generated were vertex-parity and parity-connectivity, was able to optimally solve in the first phase 55 instances, instead of the 94 instances that are optimally solved when the exact separation algorithm for the general case is also applied.

However, the increase on the cpu-time due to the application of the exact separation procedure is quite small, and the efficiency of the algorithm can be appreciated in the required cpu-times. The distribution of the cpu-times needed to solve the LP relaxation is the following: 36 instances required less than 1 second; for 38 instances the cpu-time was in the interval $[1, 10)$; for 18 instances the cpu-time was in the interval $[10, 60)$; for 7 instances the cpu-time was in the interval $[60, 120)$. Nineteen instances required more than 120 seconds of cpu-time, of which six required more than 600 seconds of cpu-time. Quite surprisingly, the cpu-time requirements of the heuristic were in quite a few cases higher than those of the iterative LP solver.

The results of the last three columns of Table 2 indicate that in most cases the considered instances could be optimally solved efficiently. Note that with our solution method the percent deviation between the upper and lower bounds at the end of the first phase is only a reliable indicator of "how far" we are from optimality for the current model. However, there might be inequalities of types (2), (3),(4) which have not been generated so far, which are necessary to characterize the optimal solution to the original problem. So, although this is not likely to happen, even if the percent gap at the end of the first phase is small it might be necessary to solve more than one integer problem. Of the 31 instances that were not solved optimally in

the first phase, 15 required only one iteration of the second phase (that is, the solution of one integer problem), 9 required two iterations, 3 required five iterations, one required 10 iterations, and the remaining instance required the solution of 13 integer problems. In our opinion, the indicator of the difficulty for solving a given instance is the number of nodes that were explored in the second phase. It is worth noting that 16 of the 31 instances that entered the second phase could be optimally solved without exploring any node of any search tree. This means that the optimal solution was found in the presolve phase of the search, which means that the cuts used by CPLEX at the root node (Gomory cuts, etc.) together with the cuts that we had generated in the first phase were enough as to characterize an optimal vertex. For the remaining instances, the total number of nodes that had to be solved during all the iterations of the second phase is smaller than 10 for seven instances, in the interval $[10, 20)$ for three instances, in the interval $[20, 100)$ for two instances, and the remaining three instances required to explore more than 100 nodes to be optimally solved. As could be expected these three instances are large ones: one in the group PD100, and two in PG100.

In general, the cpu-times required for the second phase of our algorithm are small. Out of the 31 instances that entered this second phase, nine required less than one second; for eight instances this time was in the interval $[1, 10)$; for six instances the cpu-time was in $[10, 60)$; the cpu-time for one instance was in $[60, 120)$; and, three more instances required a cpu-time in the interval $[120, 160)$. Only four instances required more than 600 seconds of cpu-time, one in PD100, one in PG64, and two in PG100. Out of these, only the instance in PD100 required more than 1,200 seconds of cpu to be optimally solved (nearly 12,000 seconds).

The results depicted in Table 2 assess the good behavior of the selected algorithm. In our opinion this is basically due to the efficiency of the first phase where in most cases a quite small subset of inequalities is sufficient for finding an optimal solution was found. In the cases when this subset was not complete, just a few additional inequalities were needed and, thus, the computational burden of the second phase was small. For the instances that entered the second phase, the average ratio between "cpu-time of the first phase" over "cpu-time of the second phase" was over 16, thus indicating the primacy of the first phase over the second one. One additional positive aspect of our algorithm is its ease of implementation.

Since in [5] it was proven that the RPP is a particular case of the PRPP, we have also performed a second series of computational experiments with RPP benchmark instances. Given that the focus of this work is the PRPP and not the RPP, the goal of these experiments is to illustrate how, in fact, of our solution algorithm for PRPP can also be used efficiently for solving specific RPP instances, and not to compete with the most effective ad hoc methods for the RPP [11, 18, 22]. To this end, we have again considered the same set of RPP benchmark instances used for the first series of experiments, that have been considered in several works for the RPP (see, for instance [11, 18, 22]). As can be seen in Table 1 their sizes go up to 100 vertices, 4950 edges and 200 R-sets, although they are not the largest RPP instances we are aware of with up to 1000 vertices, 3000 edges and 200 R-sets (see Corberán, Plana and Sanchis [13]). From [5] we know that we can transform RPP instances into equivalent PRPP instances by i) keeping in PRPP the cost function c of RPP, and ii) by defining a profit function b in PRPP that assigns high enough profits to the edges that are required in RPP and zero profit for the edges that are non-required in RPP. According to this, for the second series of experiments we have set

$b_e = 1000c_e, \forall e \in E_R$, and $b_e = 0, \forall e \in E \setminus E_R$.

The results with this second set of instances are summarized in Table 3. The meaning of the entries is the same as before. As can be seen the obtained results are very good. Despite the fact that neither the model that we use nor our algorithm exploit explicitly the structure of RPP instances, the results that we have obtained are nearly as good in terms of both the quality of the obtained solutions and the cpu-time requirements as the best published results for the same sets of benchmark instances [11, 18, 22]. In particular, the heuristic provided the optimal solution for 117 of the 118 instances. We attribute the good behavior of the heuristic to the fact that it is, in fact, an adaptation of a heuristic that has been designed for RPP instances. In 76 of the instances, we could prove the optimality of the heuristic solution because the value of the LP relaxation at the end of the first phase was optimal (0% percent gap between the upper and the lower bounds). For the remaining 42 instances the percent gaps between the upper bound given by the LP relaxation and the optimal value was always smaller than 0.015%. These percent gaps are really small, even if we take into account that the value of the objective function is very big, given that the profit function takes very big values (the largest percent gap, 0.0146%, corresponds to an absolute gap between the upper and lower bounds of 49; the largest absolute gap is 228, corresponding to a 0.0015% percent gap). In fact, for another 8 instances that were not optimally solved by the LP relaxation, the upper bound also allowed to prove the optimality of the heuristic solution. In general, the number of LP iterations of the algorithm is small. This number is smaller than 20 for 76 instances, whereas 12 instances required more than 100 iterations to terminate. As with the former set of instances, the instances that required more iterations are usually the ones with a larger number of connected components in the graph induced by the required edges of RPP. These are the instances that also tend to be harder to solve as RPP instances too. Again, parity-connectivity cuts (11), followed by connectivity cuts (3)-(4) are more frequent than node-parity cuts (12) and general co-circuit cuts (2). We have observed that applying the exact separation algorithm for the general co-circuit inequalities also improves considerably the performance of the algorithm for this set of instances. The previous version where, among co-circuit inequalities, only vertex-parity and parity-connectivity inequalities were generated was able to solve optimally in the first phase 31 instances, instead of the 76 instances that are optimally solved when the exact separation algorithm for the general case is also applied.

Like for the PRPP instances the cpu-times required to solve the LP relaxation are small for instances of this difficulty. The distribution of the cpu-times needed to solve the LP relaxation is the following: 49 instances required less than 1 second; for 32 instances the cpu-time was in $[1, 10)$; for 12 instances the cpu-time was in $[10, 60)$; for 1 instance the cpu-time was in $[60, 120)$. The 21 remaining instances required more than 120 seconds of cpu-time, with two pathological instances (one in D100 and another one in G100) that required more than 10,000 seconds of cpu-time. Again, like with the PRPP instances, the cpu-time requirements of the heuristic were in quite a few cases higher than those of the iterative LP solver.

In general, the effort required by the second phase and, hence, to optimally solve the instances, is higher than with the PRPP instances. We attribute this to the fact that, as already mentioned, we are not exploiting the structure inherent to the RPP and we are artificially transforming it into a different problem. Despite this these times are usually small if we take into account the difficulty of the considered instances. In particular, only four instances required

more than 1,200 seconds of cputime. The maximum number of iterations of the second phase was 19, and it was required by one instance in G100. None of the remaining 33 instances that entered the second phase required more than 8 iterations. As for the number of nodes explored in the second phase, 10 of the 34 instances that entered the second phase were optimally solved without exploring any node of the search tree (only with the cuts that we generated plus the cuts generated with the CPLEX presolve). For the remaining 24 instances this number of nodes was smaller than 10 for 10 instances, in the interval $[10, 20)$ for 4 instances, in the interval $[20, 100)$ for four more instances, and it was over 100 for the remaining five instances (one in D64, two in D100 and two in G100).

Before concluding, we will like to mention that we also tried to use penalties and tests for fixing the values of some variables or for reinforcing some of the existing inequalities. All the penalties and the tests that we applied are straightforward applications of well known sensitivity analysis for linear programming. Since we did not observe any significative difference on the results that we obtained when applying such procedures, we do not report them here.

8 Conclusions

In this work we have presented an algorithm for solving the Prize-collecting Rural Postman Problem that was introduced in [5]. The algorithm consists of two phases. The first one provides upper and lower bounds. Upper bounds are obtained with an iterative LP-based cutting plane algorithm and lower bounds are obtained with a heuristic (which is an adaptation of a heuristic for RPP) that takes as starting point the solution of the last LP relaxation. We have presented procedures to solve exactly the separation problem for the generated cuts. In particular we propose a new algorithm for the exact separation of general co-circuit inequalities, which is simpler and faster than the one proposed by Grötschel and Holland [24]. The second phase of the algorithm resorts to the solution of integer problems that are iteratively reinforced with the addition of cutting planes.

In order to evaluate the performance of the algorithm we have solved exactly two sets of instances with 118 instances each of them. For the first set of PRPP instances the first phase of the algorithm has given very tight upper and lower bounds in small computation times. For 94 instances the LP value was already the optimal value and only for two instances de percent gap was above 5%. The additional effort to solve exactly the instances was also quite small, since only four instances required more than 600 seconds of cpu-time in the second phase. The second set of instances consists of RPP instances transformed into equivalent PRPP instances. For these instances the behavior of the heuristic is remarkable since it gave an optimal solution in 117 instances, and the percent gap at the end of the first phase of the algorithm was always smaller than 0.015%. Thus, we can conclude that the results provided by the algorithm are very satisfactory.

9 Acknowledgements

The authors are grateful to three anonymous referees for their valuable comments and suggestions that indeed have led to an improvement on the paper. This research has been partially supported by grant MTM2006-14961-C05-01 of the Inter-Ministerial Spanish Commission of Science and Technology. The research of the first author has been partially financed by the Spanish "Secretaría de Estado de Educación y Universidades". These supports are gratefully acknowledged.

References

[1] J. Aráoz , W. Cunningham , J. Edmonds and J. Green–Krotki. Reductions to 1–matching polyhedra. *Networks* 13, 455–473, 1983.

[2] J. Aráoz, E. Fernández, C. Franquesa and O. Meza. Prize-collecting Arc Routing Problems and Extensions. Third International Workshop on Freight Transportation and Logistics (Odysseus 2006), Altea (Spain), 2006.

[3] J. Aráoz, E. Fernández and O. Meza. An LP based algorithm for the Privatized Rural Postman Problem. Research Report DR-2005/10, EIO Departament, Technical University of Catalonia (Spain), 2002.

[4] 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). $http://www.optimization-online.org/DB_HTML/2007/11/1827.html2007$.

[5] J. Aráoz, E. Fernández and C. Zoltan. The Privatizad Rural Postman Problem. *Computers and Operations Research* 33, 3432-3449, 2006.

[6] F. Barahona and M. Groetschel. On the cycle polytope of a binary matroid. *Journal of Combinatorial Theory* 40,40–62, 1986.

[7] F. Barahona and A.R. Mahjoub. On the cut polytope. {Mathematical Programming} 36,157-163, 1986.

[8] J. M. Belenguer and E. Benavent. The capacitated arc routing problem: valid inequalities and facets. *Computational Optimization and Applications* 10, 165-187, 1998.

[9] E. Benavent, A. Corberán, and J.M. Sanchis. Linear programming based methods for solving arc routing problems. In *Arc Routing: Theory, Solutions and Applications* M. Dror (edt), Kluwer Academic Publishers, 2000.

[10] 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.

[11] A. Corberán, A. Letchford and J. M. Sanchis. A cutting plane algorithm for the General Routing Problem, *Mathematical Programming*, 90, 291-316, 2001.

[12] E. Benavent, A. Corberán, J.M. Sanchis and I. Plana. Min-Max K-vehicles Windy Rural Postman Problem, Technical Report TR09-2007, Statistics and Operations Department, University of Valencia (Spain), http://www.uv.es/sestio/TechRep/tr09-07.pdf, 2007.

[13] A. Corberán, I. Plana and J. M. Sanchis. A Branch and Cut Algorithm for the Windy General Routing Problem and Special Cases. *Networks* 49(4), 245-257, 2007.

[14] A. Corberán and J. M. Sanchis. A polyhedral approach to the rural postman problem. *European Journal of Operatinal Research* 79, 95-114, 1994.

[15] A. Corberán and J. M. Sanchis. The General Routing Problem: facets from the RPP and GTSP polyhedra. *European Journal of Operational Research* 108, 538-550, 1998.

[16] 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. *European Journal of Operational Research* 127(1), 69-77, 2000.

[17] M. Dror (edt). *Arc Routing: Theory, Solutions and Applications*. Kluwer Academic Publishers, 2000.

[18] E. Fernández, O. Meza, R. Garfinkel, and M. Ortega. On the undirected rural postman problem: Tight bounds based on a new formulation. *Operations Research* 51, 281–291, 2003.

[19] D. Feillet, P. Dejax and M. Gendreau. The profitable Arc Tour Problem: Solution with a Branch-and-Price Algorithm. *Transportation Science* 39(4), 539–552, 2005.

[20] R.E. Gomory. Outline of an Algorithm for Integer Solutions to Linear Pograms. *Bulletin of the American Mathematical Society* 64, 275-278, 1958.

[21] T.C. Hu, Combinatorial Algorithms, Addison-Wesley, 1982.

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

[23] R.E. Gomory and T.C. Hu. Multi-terminal network flows. *SIAM Journal of Applied Mathematics* 9, 551-556, 1961.

[24] M. Grötschel and O. Holland. A Cutting Plane Algorithm for Minimum Perfect 2-Matchings. *Computing* 39, 327–344, 1987.

[25] A. G. Hertz, P. Laporte and H. Nanchen. Improvement procedures for the undirected rural postman problem. *INFORMS Journal on Computing* 1, 53-62, 1999.

[26] M. Padberg and M.R. Rao. Odd minimum cut-sets and b-matchings. *Mathematics of Operations Research* 7, 67-80, 1982.

[27] ILOG, *Using the CPLEX Callable Library*, 7th. edition, IOLG Inc. CPLEX Division, Incline Village, 2000.

	$\#inst.$	$\|V\|$	$\|E\|$	$\|R\|$	$\|P\|$
ALBAIDAA	1	102	5151	160	10
ALBAIDAB	1	90	4005	144	11
P	24	7-50	21-1225	13-184	2-8
D16	9	16	120	31-32	2-5
D36	9	36	630	72	4-11
D64	9	64	2016	128	5-15
D100	9	100	4950	200	9-22
G16	9	16	120	24	3-5
G36	9	36	630	60	5-9
G64	9	64	2016	112	4-14
G100	9	100	4950	180	4-20
R20	5	20	190	37-75	3-4
R30	5	30	435	70-111	4-6
R40	5	40	780	82-203	5-9
R50	5	50	1225	130-203	7-12

Table 1: Instances summary

	gap_{RL}	$\#opt_{RL}$	t_{RL}	gap_H	$\#opt_H$	t_H	$\#LP$	$v-par$	$par-conn$	$S-par$	$conn.$	$\#IP$	$\#nod$	t_{IP}
PALBAIDAA	0.19	0/1	562.50	0.30	0/1	471.65	89.00	52.00	134.00	6.00	92.00	1.00	0.00	27.197
PALBAIDAB	0.00	1/1	25.15	0.00	1/1	102.59	9.00	38.00	56.00	2.00	38.00	0.00	0.00	0.000
PP	0.44	20/24	1.97	2.05	17/24	4.05	5.96	13.79	9.00	1.92	11.63	0.75	0.63	0.719
PD16	0.51	8/9	0.30	0.00	9/9	0.41	4.89	5.22	7.78	1.22	6.78	0.22	0.22	0.082
PD36	0.12	5/9	14.58	1.47	6/9	23.35	44.78	13.89	48.44	5.78	21.78	0.56	0.22	0.578
PD64	0.14	7/9	105.34	0.37	7/9	56.65	55.78	30.89	107.33	13.56	84.89	0.33	0.89	1.706
PD100	0.40	4/9	1890.71	2.98	4/9	746.03	269.22	53.56	265.33	15.00	210.22	1.78	127.78	1379.572
PG16	0.00	9/9	0.28	0.00	9/9	0.30	4.44	1.00	8.56	0.00	1.67	0.00	0.00	0.000
PG36	0.00	9/9	18.31	0.00	9/9	27.74	46.78	11.22	54.78	3.11	14.56	0.00	0.00	0.000
PG64	1.85	8/9	139.97	0.35	8/9	121.63	62.44	21.33	197.33	0.78	30.22	0.33	7.11	96.796
PG100	0.44	6/9	2798.35	1.79	5/9	1097.41	205.22	41.00	345.56	17.22	189.67	2.22	134.67	281.379
PR20	0.26	4/5	0.40	2.35	4/5	0.37	7.50	4.00	18.50	0.00	1.75	0.25	0.00	0.010
PR30	0.00	5/5	2.70	0.00	5/5	2.01	18.40	7.60	75.00	0.00	5.00	0.00	0.00	0.000
PR40	0.00	4/5	3.69	1.23	4/5	2.04	15.40	8.60	65.60	1.20	9.40	0.20	0.00	0.114
PR50	0.07	4/5	60.86	1.29	4/5	16.37	82.60	11.80	80.20	1.20	10.00	0.20	0.00	0.355

Table 2: Results for PRPP instances

	gap_{RL}	$\#opt_{RL}$	t_{RL}	gap_H	$\#opt_H$	t_H	$\#LP$	$v-par$	$par-conn$	$S-par$	$conn.$	$\#IP$	$\#nod$	t_{IP}
ALBAIDAA	0.0002	0/1	45.09	0.00000	1/1	138.55	36.00	53.00	23.00	38.00	180.00	0.00	0.00	0.00
ALBAIDAB	0.0004	0/1	44.63	0.00000	1/1	88.30	14.00	59.00	26.00	8.00	40.00	2.00	0.00	1.25
P	0.0010	13/24	0.57	0.00000	24/24	3.93	6.25	13.29	2.67	2.75	24.00	0.92	1.38	0.28
D16	0.0044	6/9	0.16	0.00000	9/9	0.74	8.33	4.78	8.44	3.44	24.67	1.22	1.33	0.06
D36	0.0005	4/9	1.63	0.00000	9/9	3.68	13.33	12.11	54.00	6.22	25.00	0.56	9.33	0.78
D64	0.0004	4/9	23.45	0.00000	9/9	17.97	52.33	30.89	99.78	7.22	56.22	1.56	38.89	3.20
D100	0.0012	2/9	2587.35	0.00005	8/9	6330.48	311.00	158.78	406.78	26.22	274.89	2.67	430.78	2342.60
G16	0.0000	9/9	0.21	0.00000	9/9	1.12	8.11	3.11	12.78	1.44	9.00	0.00	0.00	0.00
G36	0.0008	7/9	2.68	0.00000	9/9	7.46	15.11	13.22	49.89	4.67	29.56	0.11	0.00	0.04
G64	0.0005	8/9	47.68	0.00000	9/9	43.86	27.33	29.11	166.56	5.89	38.78	0.11	0.78	1.95
G100	0.0010	4/9	2827.84	0.00000	9/9	3237.19	241.56	68.67	404.33	19.11	229.33	3.44	344.44	1567.78
R20	0.0000	5/5	0.55	0.00000	5/5	0.45	10.50	1.50	29.00	0.25	4.00	0.00	0.00	0.00
R30	0.0000	5/5	3.89	0.00000	5/5	3.63	29.80	4.80	121.40	0.40	5.00	0.00	0.00	0.00
R40	0.0003	4/5	53.34	0.00000	5/5	24.66	95.20	5.20	135.60	5.40	39.20	0.20	0.00	0.34
R50	0.0000	5/5	191.25	0.00000	5/5	155.74	276.20	8.40	172.60	2.40	17.80	0.00	0.00	0.00

Table 3: Results for RPP instances