A Cutting Plane Algorithm for the General Routing Problem.

Angel Corberán°, Adam N. Letchford* and José María Sanchis+

° *DEIO, Faculty of Mathematics, University of Valencia, Spain*

* *Dept. of Management Science, Lancaster University, England*

+ *Dept. of Applied Mathematics, University Polytechnic of Valencia, Spain*

November 9th, 1998.

Abstract

The *General Routing Problem* (GRP) is the problem of finding a minimum cost route for a single vehicle, subject to the condition that the vehicle visits certain vertices and edges of a network. It contains the *Rural Postman Problem*, *Chinese Postman Problem* and *Graphical Travelling Salesman Problem* as special cases. We describe a cutting plane algorithm for the GRP based on facet-inducing inequalities and show that it is capable of providing very strong lower bounds and, in most cases, optimal solutions.

Key Words: valid inequalities, cutting planes, General Routing Problem, Rural Postman Problem, Graphical Travelling Salesman Problem.

1 Introduction

Given a connected, undirected graph G with vertex set V and edge set E, a cost c_e for each edge $e \in E$, a set $V_R \subseteq V$ of *required vertices* and a set $E_R \subseteq E$ of *required edges*, the *General Routing Problem* (GRP) is the problem of finding a minimum cost vehicle route passing through each $v \in V_R$ and each $e \in E_R$ at least once [24].

The GRP is of practical application (see, e.g., [9]) and contains several other important routing problems as special cases:

- When $V_R = \emptyset$, the *Rural Postman Problem* (RPP) is obtained [24]. If, in addition, $E_R = E$, the *Chinese Postman Problem* (CPP) is obtained [8].
- When $E_R = \emptyset$, the *Steiner Graphical Travelling Salesman Problem* (SGTSP) is obtained [7]. This problem was also called the *Road Travelling Salesman Problem* in [10]. If, in addition, $V_R = V$, the *Graphical Travelling Salesman Problem* (GTSP) is obtained [7].

The RPP was shown to be $\mathcal{NP}$-hard in [18], although the CPP can be solved in polynomial time by reduction to a matching problem [8]. The GTSP is also $\mathcal{NP}$-hard [7]. Hence the SGTSP and GRP are also $\mathcal{NP}$-Hard.

The GTSP is a relaxation of the well-known *(Symmetric) Travelling Salesman Problem* (STSP): in the GTSP, the route must pass through each vertex at least once and each edge may be traversed any number of times. In the STSP, the route must pass through each vertex *exactly* once and, moreover, G must be a complete graph (see, e.g., [16, 17]).

In recent years, spectacular progress has been made on solving large-scale STSP instances to optimality. The most successful algorithms to date (e.g., [2, 27]) are based on linear programming relaxations, strengthened by the addition of facet-defining inequalities as cutting planes (see, e.g., [22]). Successful algorithms have also been devised for other routing problems using this paradigm [2, 3, 10, 11, 14, 23].

An important ingredient of a cutting plane algorithm is the ability to detect inequalities which are violated by the current LP relaxation. For a given class of inequalities, an *exact separation algorithm* is a routine which takes an LP relaxation as input and outputs one or more violated inequalities in that class (if any exist). A *heuristic separation algorithm* is similar except that it may fail to detect a violated inequality in the class (see, e.g., [26]).

In [5], various facet-inducing inequalities were discovered for the RPP, including *non-negativity*, *connectivity*, *R-odd cut* and *K-Component* (K-C) inequalities. Using these inequalities, the authors solved 25 out of 26 RPP instances using a pure cutting plane algorithm (the other was solved by branching). However, the violated inequalities were identified by visual inspection rather than by automated separation algorithms.

The present authors have also written three papers on GRP polyhedra. The non-negativity, connectivity, R-odd cut and K-C inequalities were adapted to the GRP in [6], where also the K-C inequalities were generalized to give the *honeycomb* inequalities. In [19], the *path-bridge* inequalities were introduced and an exact separation algorithm was proposed for a restricted subclass. Further classes of inequalities were introduced in [20], including *projected binested*, *projected chain* and *projected bipartition* inequalities.

These good theoretical results suggested that it would be possible to solve large-scale GRP instances to optimality, or at least to obtain excellent lower bounds, with a dedicated cutting-plane algorithm. The purpose of the present paper is to give a description of such an algorithm which has been implemented and tested by the authors. This includes a detailed description of some exact and heuristic (polynomial-time) separation algorithms which have been devised for various classes of valid and facet-inducing inequalities.

Note that it can be assumed, without loss of generality, that the end-vertices of any required edge are also required. Our algorithms actually rely on the stronger assumption that $V = V_R$. This is not a serious restriction, however, as there is a simple way to transform GRP instances which do not satisfy the assumption into instances which do (see, e.g., [4]). The transformation can occasionally increase the number of edges in G, but it frequently decreases it.

In [16], some RPP instances were solved by transforming them into STSP instances and then using a branch-and-cut algorithm for the STSP. Although this worked adequately for the instances tested, the transformation involved the addition of a large number of redundant variables. Our algorithm is more 'natural' and has some dedicated separation algorithms which work very well. We believe that it will be much faster for larger instances.

The outline of the paper is as follows. In Section 2, the relevant results on GRP polyhedra are reviewed. In Section 3, a detailed description is given of the separation algorithms. The way in which these separation algorithms are integrated to form the overall algorithm is described in Section 4. Computational results are given for a wide variety of test problems in Section 5. Some concluding comments are made in Section 6.

2 Polyhedral Results

2.1 Formulation

The fundamentals of the integer programming approach to the GRP are given in [6]. We use the same notation, with some minor simplifications due to our assumption that $V = V_R$. Let x_e represent the number of times e is traversed (if $e \notin E_R$), or one less than this number (if $e \in E_R$). Given $S \subset V$, let $\delta(S)$ denote the set of edges, commonly called the *edge cutset*, with one end-vertex in S and one end-vertex in $V \setminus S$. We also write $E(S : T)$ for $\delta(S) \cap \delta(T)$, $E(S)$ for the set of edges with both end-vertices in S and $x(F)$ for $\sum_{e \in F} x_e$. Finally, we write $E_R(S : T)$ for $E(S : T) \cap E_R$ and $\delta_R(S)$ for $\delta(S) \cap E_R$. The set of feasible solutions to the GRP is then described by:

$$x(\delta(S)) \geq 2, \qquad \forall S \subset V, \delta_R(S) = \emptyset \tag{1}$$

$$x(\delta(\{i\})) \equiv \delta_R(\{i\}) \bmod 2, \qquad \forall i \in V \tag{2}$$

$$x \in Z_+^{|E|} \tag{3}$$

The *connectivity* inequalities (1) ensure that the route is connected. The *degree conditions* (2) ensure that the vehicle departs each vertex as many times as it arrives. Note that the degree conditions are congruences, not linear equations or inequalities.

The convex hull in $\Re^{|E|}$ of feasible solutions to (1) - (3), known as GRP(G), is a full-dimensional, unbounded polyhedron. We can therefore formulate the GRP as the problem of minimizing $\sum_{e \in E} c_e x_e$ subject to $x \in$ GRP(G). As mentioned in Section 1, many classes of valid inequalities and facets are known for GRP(G) [5, 6, 19, 20]. The most trivial inequalities are the *non-negativity* inequalities $x_e \geq 0$ for each $e \in E$. These induce facets unless e is a cut-edge in G.

The non-negativity inequalities are handled implicitly by any LP solver. The authors have devised separation algorithms for five *non-trivial* classes of inequalities; namely, *connectivity*, *R-odd cut*, *K-C*, *path-bridge* and *honeycomb* inequalities. These inequalities are described in the following subsections.

2.2 Connectivity inequalities

As mentioned in the previous subsection, connectivity inequalities are just constraints (1). They induce facets of GRP(G) if and only if the subgraphs induced by S and $V \setminus S$ are connected [5, 6].

2.3 R-odd cut inequalities

Due to the degree conditions, the vehicle must cross any given edge cutset an even number of times. This means that if $S \subset V$ is such that $|\delta_R(S)|$ is odd, then the *R-odd cut* inequality

$$x(\delta(S)) \geq 1 \tag{4}$$

is valid for GRP(G).

Like connectivity inequalities, R-odd cut inequalities induce facets of GRP(G) if and only if the subgraphs induced by S and $V \setminus S$ are connected [5, 6].

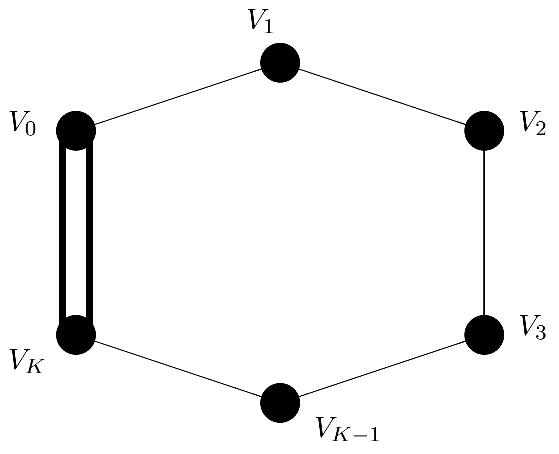


Figure 1: K-C configuration.

2.4 K-C inequalities

In order to present the remaining inequalities, we will need some more definitions. Consider the (generally disconnected) subgraph of G obtained by deleting all non-required edges from G. We call a connected component of this subgraph an *R-connected component.* A set of vertices defining an R-connected component will be called an *R-set.* An R-set with only one member will be called an *isolated vertex.*

The *K-Component* or *K-C* inequalities [5, 6] are defined in terms of an associated *K-C configuration.* A K-C configuration is (see Figure 1) a partition $\{V_0, \dots V_K\}$ of V, with $K \geq 3$, such that

- $V_1, \dots V_{K-1}$ and $V_0 \cup V_K$ are clusters of one or more R-sets,
- $|E_R(V_0 \ : \ V_K)|$ is positive and even,
- $E(V_i \ : \ V_{i+1}) \neq \emptyset$ for $i \ = \ 0, \dots, K-1$.

The corresponding K-C inequality can be written as:

$$F(x) = \sum_{i=0}^{K-1} x(\delta(V_0 \cup \cdots \cup V_i)) - 2x(E(V_0 : V_K)) \ \geq 2(K-1) \tag{5}$$

K-C inequalities induce facets of GRP(G) when certain mild connectivity assumptions are met [5, 6].

2.5 Path-bridge inequalities

The *path-bridge* (PB) inequalities [19] are defined in terms of an associated *path-bridge* (PB) *configuration.* Suppose $p \geq 1$ and $b \geq 0$ are integers such that $p + b \geq 3$ and odd. Let $n_i \geq 2$ for $i = 1, \dots, p$ also be integers. A PB configuration is (see Figure 2) a partition of V into vertex sets A, Z and V_j^i for $i = 1, \dots, p$, $j = 1, \dots, n_i$ with the following properties:

- each V_j^i is a cluster of one or more R-sets,

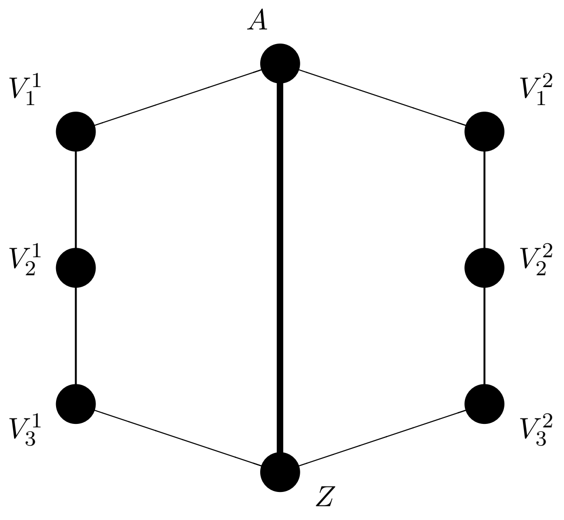


Figure 2: PB configuration.

- $|E_R(A : Z)| = b$,
- $E(A : V_1^i) \neq \emptyset$ and $E(V_{n_i}^i : Z) \neq \emptyset$ for $i = 1, \ldots, p$,
- $E(V_j^i : V_{j+1}^i) \neq \emptyset$ for $i = 1, \ldots, p$ and $j = 1, ..., n_i - 1$.

The edges in $E_R(A : Z)$ constitute the *bridge*. When $b = 0$, the bridge is empty and the PB configuration reduces to a *path configuration*, see [7]. In such a case, either or both of A and Z are permitted to be empty.

The formula for the coefficients in the PB inequality is rather complicated. However, our separation algorithms (Subsections 3.5 and 3.6) are designed for so-called *n-regular* PB inequalities, in which all of the n_i are equal to the same value n [7, 19]. For such inequalities, there is a nice description of the coefficients in terms of vertex sets called *handles* and *teeth*. There are $n - 1$ handles, denoted by $H_1, \ldots, H_{n-1}$, and p teeth, denoted by $T_1, \ldots, T_p$ (see Figure 3). The first handle is defined as $H_1 = A \cup V_1^1 \cup \ldots \cup V_1^p$; the other handles are defined inductively as $H_i = H_{i-1} \cup V_i^1 \cup \ldots \cup V_i^p$. The teeth are defined as $T_j = V_1^j \cup \ldots \cup V_n^j$. The n-regular PB inequality is then:

$$\sum_{i=1}^{n-1} x(\delta(H_i)) + \sum_{j=1}^{p} x(\delta(T_j)) \ \geq \ np + n + p - 1 \qquad (6)$$

and is valid for GRP(G). It is facet-inducing under mild connectivity assumptions.

For brevity, we will call regular PB inequalities *n-PB* inequalities (where n may or may not be specified). Certain special cases of n-PB inequalities are of note. When $p = 1$, a K-C inequality is obtained. When $b = 0$, a *regular path* inequality is obtained [7]. The 2-PB inequalities are analogous to the well-known *comb* inequalities for the STSP (see, e.g., [13]). Finally, a 2-PB inequality in which each tooth consists merely of two isolated vertices (connected by a non-required edge) is called *simple* [19]. Simple 2-PB inequalities are analogous to the *2-matching* inequalities for the STSP [13].

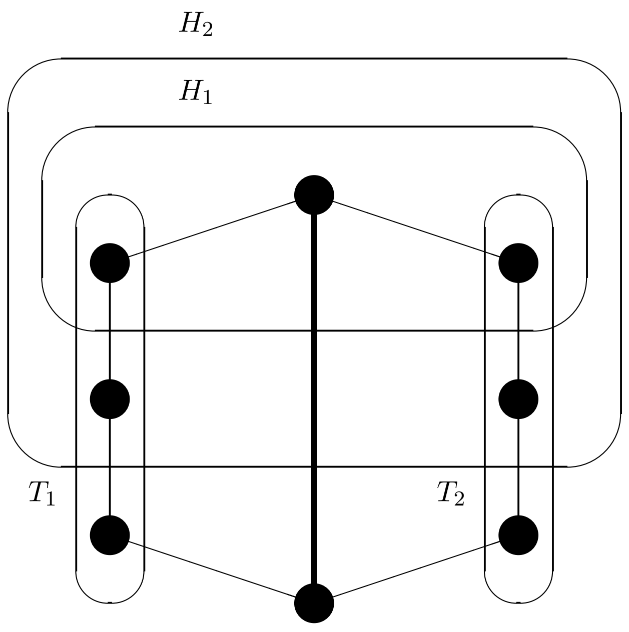


Figure 3: Handles and teeth in a 3-regular PB configuration.

2.6 Honeycomb inequalities

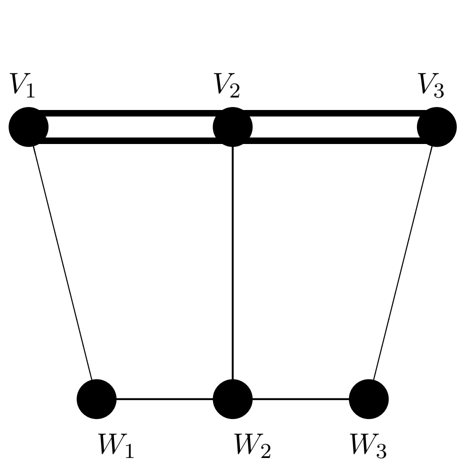
The *honeycomb* inequalities [6], like PB inequalities, are also a generalization of K-C inequalities. However, the generalization is in a different direction and neither class contains the other. A *honeycomb configuration* is a partition of V into sets S_i such that:

- for all i, $\mid \delta(S_i) \setminus \delta_R(S_i) \mid \neq \emptyset$ and $\mid \delta_R(S_i) \mid$ is even or zero;
- there are at least two values i such that $\delta_R(S_i) \neq \emptyset$;
- there are at least two values i such that $\delta_R(S_i) = \emptyset$;

together with a set of non-required edges crossing between the S_i which form a tree spanning the S_i.

Many, but not all, honeycomb configurations can be formed by 'gluing' K-C configurations together by identifying edges [6]. In general, honeycomb configurations can be extremely complicated and computing the coefficients in the associated honeycomb inequality can be a formidable task. For this reason, we restrict ourselves in this paper to honeycomb configurations with a special structure. These consist of:

- a partition $\{V_1, \ldots, V_L, W_1, \ldots, W_{K-1}\}$ of V, with $L \geq 2$, $K \geq 3$, such that $(V_1 \cup \ldots \cup V_L)$, $W_1, \ldots, W_{K-1}$ are clusters of one or more R-sets, $\delta(V_i)$ contains a positive and even number of required edges for all i and the graph induced by the required edges on the vertex set $\{V_1, \ldots, V_L\}$ is connected.
- a tree T spanning the sets $V_1, \ldots, V_L, W_1, \ldots, W_{K-1}$ such that the degree in T of every vertex set V_i is 1, the degree of vertex sets W_j is at least 2 and the path in the tree connecting any distinct V_i, V_j is of length 3 or more.



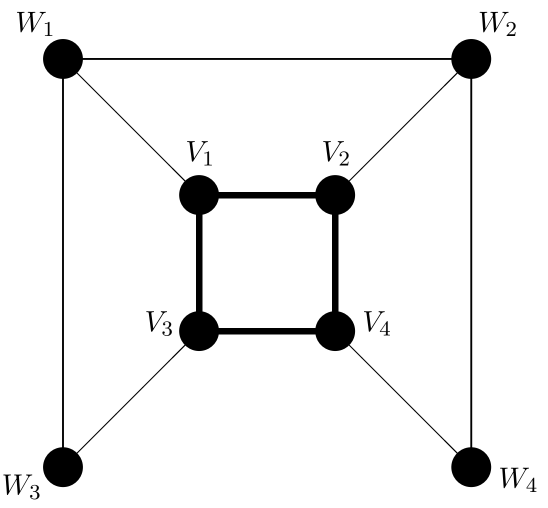


Figure 4: Two suitable honeycomb configurations.

If $L = 2$ the tree degenerates to a mere path and we have a K-C configuration. If $L \geq 3$, then $K \geq 4$ is needed in order for the path in the tree connecting any distinct V_i, V_j to be of length 3 or more.

For honeycomb configurations with this special structure, the coefficient α_e of edge $e \in E$ in the associated honeycomb inequality is equal to the number of edges traversed (if any) in the spanning tree to get from one end-vertex of e to the other, except for the edges with one end-vertex in V_i and the other in V_j, $i \neq j$, when the coefficient is 2 units less. The honeycomb inequality is then:

$$\sum_{e \in E} \alpha_e x_e \geq 2(K-1) \tag{7}$$

These honeycomb inequalities define facets of GRP(G) if certain mild connectivity assumptions are met.

Figure 4 shows two suitable honeycomb configurations. The bold lines represent edges in $\delta_R(V_i)$ for some i and the thin lines represent edges in the spanning tree. The rhs of the associated inequality is 6 for the configuration on the left and 8 for the configuration on the right.

3 Separation Algorithms

3.1 Preliminary Comments

In this section, separation algorithms are presented for the inequalities described in the previous section. Before proceeding, however, we will need some definitions.

Given an LP relaxation vector $x^* \in \Re_+^{|E|}$, define the weighted graph $G_*(V, E, x^*)$. This weighted graph is input to all of the separation procedures. Given a graph $G(V, E)$, a *cut-vertex* is a vertex the removal of which causes G to become disconnected. A *block* is a maximal connected subgraph of G which contains no cut-vertices. If a block consists of a single edge, then it is called a *cut-edge.* A decomposition of G into blocks can be found in

$\mathcal{O}(|E|)$ time using the algorithm in [29]. We will also need the concept of *shrinking* a set of vertices in a weighted graph (see, e.g., [10, 26]). Given a graph $G(V, E)$ with weights on the edges, and a set $W \subset V$, shrinking W means identifying the vertices in W, deleting any resulting loops and merging each resulting set of parallel edges, if any, into a single edge. When merging parallel edges, the new edge is given a weight equal to the sum of the original weights. Shrinking can be done iteratively and, in the case of the GRP, a single edge in the shrunk graph can represent a mixture of any number of required or non-required edges in the original graph.

In the following subsections, we describe heuristic (and sometimes exact) separation algorithms for each of the classes of inequalities described in the previous section. Some of the heuristics require the choice of a parameter (called ϵ), or even two parameters (called ϵ_1, ϵ_2). For details on which values of ϵ (or ϵ_1, ϵ_2) we use and how the exact and heuristic algorithms are embedded in the overall algorithm, see Subsection 4.2.

3.2 Connectivity Separation

Connectivity inequalities can be separated exactly in polynomial time. This is done by finding a minimum weight cut in the shrunk graph $G_s = (V_s, E_s, \bar{x}^*)$ obtained from G_* by shrinking each R-set into a single vertex. A minimum weight cut in an undirected graph with n vertices and m edges can be found in $\mathcal{O}(nm + n^2 \log n)$ time using the algorithm in [21].

A faster ($\mathcal{O}(|E|)$ time) heuristic algorithm is obtained by computing the connected components of the subgraph induced by the edges $e \in E_s$ with $\bar{x}^*_e > \epsilon$, where ϵ is a given parameter. Let $S_1, S_2, \ldots, S_q$ be the sets of vertices in the original graph G corresponding to the vertex sets of these connected components. Then $x(\delta(S_i)) \geq 2$ is a violated connectivity inequality if $q > 1$ and $x^*(\delta(S_i)) < 2$. Note that when $q = 2$ we have $x(\delta(S_1)) = x(\delta(S_2))$, but when $q > 2$ all of the inequalities are distinct.

3.3 R-odd cut separation

R-odd cut inequalities can also be separated exactly in polynomial time. Before describing the exact algorithm, however, we describe an effective separation heuristic. As for the connectivity inequalities, we choose a parameter ϵ. We compute the vertex sets $S_1, S_2, \ldots, S_q$ of the connected components of the subgraph of G_* induced by the edges $e \in E$ with $x^*_e > \epsilon$. Then $x(\delta(S_i)) \geq 1$ is a violated R-odd cut inequality if $q > 1$, $|\delta_R(S_i)|$ is odd and $x^*(\delta(S_i)) < 1$.

This heuristic runs in $\mathcal{O}(|E|)$ time and is inspired by a heuristic presented in [12] for *blossom* inequalities in the context of the perfect matching problem.

The exact algorithm is much slower and involves finding a minimum weight R-odd cutset in G_*. Using a result of [25], this can be reduced to a series of maximum flow problems on G_*. The number of maximum flow computations needed in the worst case is equal to the number of R-odd vertices in G minus 1 and an R-odd cut inequality is violated if and only if the weight of the cutset is less than 1.

As noted in [26] (in the context of the STSP), the idea of connected components can also be used to simplify the problem of finding an odd cut of weight less than 1. Consider the connected components of the subgraph of G_* induced by the edges $e \in E$ with $x^*_e > 0$. Under the assumption that the separation heuristic has already been (unsuccessfully) invoked with $\epsilon = 0$, we know that each of these components contains an even number of R-odd vertices. It is easy to show that we can examine each of these components independently for an odd cut of weight less than 1. This typically leads to considerable speed improvements in the exact algorithm.

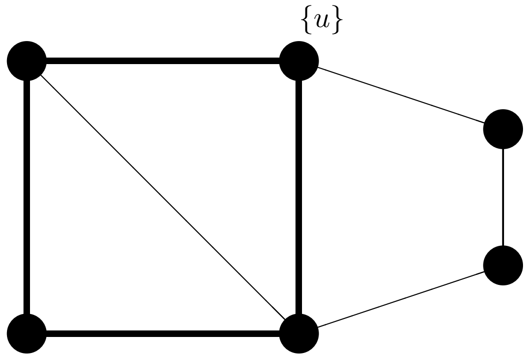


Figure 5: Getting seeds for V_0 and V_K.

3.4 K-C separation

A vertex $v \in V$ will be called *R-odd* if $|\delta_R(\{v\})|$ is odd, otherwise it will be called *R-even.* Isolated vertices are R-even.

It is not known if the problem of separating K-C inequalities can be solved in polynomial time or not, but our guess is that this problem is $\mathcal{NP}$-hard. However, we have designed a heuristic algorithm which appears to work very well. It has three consecutive phases. In phase I, we find 'seeds' for V_0 and V_K. In phase II, we use these seeds to determine the V_i for $i = 0, \ldots, K$. In phase III, we check the resulting inequality for violation, and, if it is not violated, we also check some other inequalities for violation which are obtained by shrinking the K-C configuration appropriately.

The details of these three phases are based on the following considerations:

- We have examined the structure of the solutions obtained when all the connectivity and R-odd cut inequalities are satisfied. The effect of the K-C inequalities is to separate solutions x^* where an R-even vertex u belonging to an R-set C_i with $|C_i| > 2$ satisfies $x^*(\delta(\{u\})) \cong 1$ and $x^*(E(\{u\} : C_i \setminus \{u\})) \cong 0$. Thus, $\{u\}$ and $C_i \setminus \{u\}$ are suitable vertex sets to be considered as seeds for V_0 and V_K, respectively, in phase I. Figure 5 illustrates this idea: the bold lines represent the required edges in $E(C_i)$ and the narrow lines represent non-required edges with $x^*_e = 1$. This idea is generalized by considering as seeds for V_0 any vertex sets with an even number of R-odd vertices forming an x^*-connected component on the subgraph induced by C_i.

- Given the seeds for V_0 and V_K, consider the graph obtained from G_* by shrinking the seeds and the remaining R-sets into a single vertex each. In order to define $V_0, \ldots, V_K$, we have to find a path in the shrunk graph connecting the seeds (preferably one with a large x^*-weight). Once this is done, we merely have to assign any vertices which are not in the path to one of the vertices in the path in order to complete phase II.

- Suppose that a K-C inequality with $K \geq 4$ is not violated. For $i = 0, 1, 2, \ldots, K-1$, let $LHS(i) = x^*(\delta(V_0 \cup \ldots \cup V_i))$. For some $1 \leq i \leq K-2$, we could merge sets V_i, V_{i+1} into a single set, yielding a new 'smaller' (K-1)-C configuration, with an associated (K-1)-C inequality. From equation (5), for a given x^*, the slack of the new inequality will be equal to that of the original inequality, plus $2 - LHS(i)$. Thus, iteratively merging sets V_i, V_{i+1} with $LHS(i) > 2$ will lead to inequalities with smaller slack, as long as $K \geq 3$ remains.

Alternatively, consider what would happen if we were to merge sets $V_0, \ldots, V_s$ into a single set for some $s \leq K-3$. The resulting change in slack would be $2s - \sum_{i=1}^{s}(LHS(i-1) + 2x^*(V_i : V_K))$. Then, if this quantity was negative, the slack would have been reduced by shrinking. A similar argument applies if we merge sets $V_t, \ldots, V_K$ for some $t \geq 3$.

It can also be shown that a K-C inequality cannot be violated if $x^*(E(V_0 : V_K)) \geq 1$ and x^* satisfies all connectivity and R-odd cut inequalities.

These are the ideas behind phase III.

In our computer code, a given K-C configuration is stored by simply labelling each vertex in the graph according to the set V_i of the K-C configuration it belongs to. Computing coefficients from these labels is a simple matter and so is the operation of merging two or more V_i into one.

Taking into account the above points, the global procedure is as follows:

- **Phase I: Define seeds for V_0 and V_K.**
  Given a specified R-set C_i, we say that a vertex in C_i is x^*-*external* if it is connected to at least one vertex in a different R-set by an edge e with $x^*_e > 0$. For each R-set C_i with at least two x^*-external vertices and connected to at least two different R-sets, do the following. Choose a parameter ϵ and construct the subgraph of $G(C_i)$ induced by edges e with $x^*_e > \epsilon$. Look for an x^*-external vertex u whose corresponding connected component in the subgraph has an even number of R-odd vertices. If one is found, the connected component is a candidate for V_0 and the complementary set in C_i is a candidate for V_K (if $V_K \neq \emptyset$).

- **Phase II: Define the vertex sets $V_0, \ldots, V_K$.**
  Given the seeds for V_0 and V_K, we proceed as follows:
  (a) construct the graph obtained from G_* by shrinking the seeds and the remaining R-sets into a single vertex each.
  (b) Compute a spanning tree by iteratively adding the edge of maximum weight not forming a cycle (and not connecting the seeds). If this is not possible (because the removal of C_i disconnects the graph), then move on to another x^*-external vertex.
  (c) Transform the tree into a path linking the seeds, by (iteratively) shrinking each non-seed vertex with degree one into its (unique) adjacent vertex. If the length of the path is only 2, then move on to another x^*-external vertex. Otherwise, the vertices of the path define the vertex sets $V_0, V_1, \ldots, V_K$. Note that the seeds may have been enlarged in this process and therefore that either or both of V_0 and V_K may contain vertices from more than one R-set.

- **Phase III: Check the K-C inequality.**
  We now have a K-C configuration like that of Figure 1.

  For each $i = 0, 1, \ldots, K-1$, compute $LHS(i) = x^*(\delta(V_0 \cup \ldots \cup V_i))$. Then, $F(x^*) = \sum_{i=0}^{K-1} LHS(i) - 2x^*(V_0, V_K)$ is the left hand side of the corresponding K-C inequality computed on x^*. If $F(x^*) < 2(K-1)$, the K-C inequality is violated.

  If the K-C inequality is *not* violated, check if a violated inequality could be obtained by shrinking the K-C configuration.

  - Shrink all pairs V_i, V_{i+1} with $LHS(i) > 2$, $1 \leq i \leq K-2$ and reduce K accordingly. If $K < 3$, stop (no violated inequality was found).

– Compute the values:

$$a = LHS(0) + 2x^*(E(V_1 : V_K)) - 2$$
$$b = LHS(K-1) + 2x^*(E(V_0 : V_{K-1})) - 2$$
$$c = a + b + 2x^*(E(V_1 : V_{K-1}))$$

corresponding to the reductions in slack which would be obtained if we shrank, respectively, the pair V_0, V_1 or the pair V_{K-1}, V_K, or both pairs simultaneously. If $\max\{a, b, c\}$ is larger than the slack, then shrink the pair V_0, V_1 and/or the pair V_{K-1}, V_K (depending on where the maximum of $\{a, b, c\}$ is reached), to obtain a violated K-C inequality (if $K \geq 3$ remains).

– If this fails, it may yet be possible to obtain a violated inequality by shrinking. Compute for every $0 \leq s, t \leq K-3$, $s+t \leq K-3$, the reduction in slack (denoted by A_{st}) which would be obtained if we shrank sets $V_0, V_1, \ldots V_s$ and sets $V_K, V_{K-1} \ldots V_{K-t}$ into single sets (this can be done quickly if the computations are performed in an intelligent way). If $A_{st} > \mathit{slack}$ for any s, t such that $K \geq 3$ would remain after shrinking, then select the s, t which gives the inequality with greatest violation.

For a fixed value of ϵ, the algorithm looks for only one violated K-C inequality associated to each R-set. For each R-set C_i, priority is given to single vertices in C_i as candidates for V_0. Only if this fails to yield a violated inequality for the given C_i are more complex candidates for V_0 considered.

A simple extension of the heuristic allows us to search for violated K-C inequalities where the required edges in $E(V_0 : V_K)$ belong to more than one R-connected component. This is done by (iteratively) joining a pair of R-sets adjacent in G_* and then applying the above procedure as if they formed a single R-set.

3.5 n-PB separation: the case $n = 2$

When considering the separation of n-PB inequalities, it is useful to give the 2-PB inequalities special attention. Accordingly, the present subsection is concerned with 2-PB inequalities, whereas general n-PB inequalities are dealt with in the following subsection.

In [19] it was shown, using ideas in [25], that the *simple* 2-PB inequalities can be separated exactly in polynomial time, provided that two conditions are met:

- x^* satisfies all non-negativity and R-odd cut inequalities,
- $x^*(\delta(e)) + x^*_e \geq 3$ holds for each $e \in E$.

The first condition is obviously no problem. As for the second condition, it is easy to show that it is satisfied whenever x^* satisfies all connectivity inequalities.

There are however two problems with the simple 2-PB separation algorithm. First, it is rather slow as it involves the computation of a minimum weight odd cut in a graph with $\mathcal{O}(|E|)$ vertices and $\mathcal{O}(|E|)$ edges. This problem can be alleviated to some extent by using the idea of connected components, as outlined in Subsection 3.3. The second problem is that there are likely to be very few violated simple 2-PB inequalities, if any, when the GRP instance has few isolated vertices. Indeed, simple 2-PB inequalities are not defined at all for RPP instances.

One way of improving the simple 2-PB separation algorithm would be to find conditions under which G_* can be *shrunk* (see Subsection 3.1) in such a way that

- A 2-PB inequality is violated in the shrunk graph if and only if a 2-PB inequality is violated by x^*,
- A violated simple 2-PB inequality in the shrunk graph can be 'expanded' into a (not necessarily simple) 2-PB inequality violated by x^*.

Although this idea has been applied successfully to related separation problems (see [10, 26]), we did not use it here because it turned out to be very difficult to devise appropriate data structures to account for the presence of required edges. A different idea was used instead.

Consider the graph G'_* obtained from G_* by deleting non-required edges e with $x^*_e = 0$. Suppose we have a *block decomposition* of G'_* (see Subsection 3.1) and that there are $h \geq 2$ blocks. Let $V(j)$ for $j = 1, \ldots, h$ denote the set of vertices in block j, let G^j for $j = 1, \ldots, h$ denote the subgraph of G'_* induced by $V(j)$ and let $x(j)$ denote the vector obtained from x^* by dropping all components apart from those corresponding to edges in $E(V(j))$. It is possible to show that a 2-PB inequality is violated by x^* if and only if there is at least one j such that a 2-PB inequality (valid for GRP(G^j)) is violated by $x(j)$.

We therefore apply the simple 2-PB algorithm to each G^j separately. If a violated simple 2-PB inequality is found for some G^j, we expand it into a (not necessarily simple) 2-PB inequality violated by x^*. This is done by viewing G^j as being obtained from G'_* by shrinking, in any order, those $V(i)$ such that $i \neq j$. This approach typically yields considerable speed improvements, because it is frequently the case that there are $h \geq 2$ blocks. Moreover, it typically leads to more violated inequalities being discovered. Even for RPP instances, where there are no isolated required vertices in the original graph G, it is possible for one or more *blocks* to contain isolated required vertices and therefore for violated simple 2-PB inequalities to be found on individual blocks.

To avoid difficulties due to rounding errors, we actually define G'_* a little differently: we delete from G_* all non-required edges e with $x^*_e < 0.01$.

3.6 n-PB separation: the general case

The separation of n-PB inequalities appears to be very difficult and is likely to be $\mathcal{NP}$-hard. However, the authors have devised a separation heuristic which is fast (it can be implemented to run in $\mathcal{O}(|V|.|E|.\log |V|)$ time) and quite effective. In fact, it has proved worthwhile to run this heuristic *before* the heuristic for 2-PB inequalities described in the previous subsection.

Like the 2-PB heuristic, the n-PB heuristic is applied to each block G^j of G_* separately and each resulting violated inequality is 'expanded' to yield an inequality violated by x^*. From now on we assume that we are working on an individual block, but for simplicity of notation we assume that there is in fact only one block, the graph G_* itself.

Given any two R-sets S_1, S_2 connected by at least one edge, let $\theta(S_1, S_2) = x^*(\delta(S_1 \cup S_2)) + x^*(E(S_1 : S_2)) - 3$. In limited experiments, adjacent pairs V^i_j and V^i_{j+1} in violated n-PB inequalities often turned out to be R-sets S_1, S_2 such that $\theta(S_1, S_2)$ was small. This fact is used in the first stage of the heuristic, which is to get a good set $\mathcal{H}$ of candidates for H_1:

- Choose two parameters $0 \leq \epsilon_1, \epsilon_2 \leq 1$.
- Initially all edges are unlabelled.

- Examine each pair S_1, S_2 of R-sets connected by at least one edge. If $\theta(S_1, S_2) \leq \epsilon_1$, then label the edges in $E(S_1)$ and $E(S_2)$ 'strong' and the edges in $E(S_1 : S_2)$ 'weak'. Store (S_1, S_2) as a 'candidate tooth'. S_1 and S_2 are the 'ends' of the candidate tooth.
- Examine the remaining unlabelled edges. Label such an edge e 'weak' if $x^*_e \leq \epsilon_2$.
- Delete all weak edges from G_*.
- Examine each connected component in the resulting graph. Let C be the set of vertices in one such component. Let $b = |\delta_R(C)|$ and let p equal the number of candidate teeth with exactly one end in C. If $p + b \geq 3$ and odd, $p \geq 1$ and the p candidate teeth mentioned are vertex-disjoint, then put C into the list $\mathcal{H}$ of candidates for H_1.

It is possible to implement this to run in $\mathcal{O}(|E|)$ time, by first of all computing $x^*(\delta(S))$ for each R-set S. Notice also that, for fixed ϵ_1, ϵ_2, $|\mathcal{H}|$ is $\mathcal{O}(|V|)$. The remainder of the heuristic is repeated for each candidate $H_1 \in \mathcal{H}$.

First, we check whether the simple 2-PB inequality with handle H_1 and the p candidate teeth is violated. If so, we store this violated inequality. Whether or not this simple 2-PB inequality is violated, we then proceed to look for a more general n-PB inequality which is violated.

To this end, we set $V_1^1, \ldots, V_1^p$ to be the ends of the p candidate teeth lying in H_1. The remainder of H_1, if any, is used to form A. The other ends of the p candidate teeth are used to form 'seeds' for $V_2^1, \ldots, V_2^p$. We then construct a shrunk graph from G_* as follows:

- Shrink the R-sets which are contained entirely in H_1 into a single vertex each, and do the same for the R-sets which are contained entirely in $V \setminus H_1$.
- Shrink the vertices in H_1 which are incident on one or more crossing required edges into a single 'inner pole' vertex (unless there are no such vertices).
- Shrink the vertices in $V \setminus H_1$ which are incident on one or more crossing required edges into a single 'outer pole' vertex (unless there are no such vertices).

Let $G_s(V_s, E_s, \bar{x}^*)$ denote the weighted shrunk graph and let H_s denote the shrunk H_1. In the next stage, G_s is used to enlarge the PB configuration. We distinguish two cases.

- **Case 1:** $b \geq 1$. Here, A is non-empty and we also have an obvious 'seed' for Z; namely, the outer pole. If $\bar{x}^*(E(A : Z)) \geq 1$, we give up. Otherwise we follow a similar idea to that used in the K-C heuristic: we iteratively add edges in order of non-increasing weight so as to build a tree in G_s with the following properties:
  - It spans all vertices in V_s apart from those representing A and the V_1^i.
  - It consists of $p + 1$ edge-disjoint subtrees.
  - Subtree i for $i = 1, \ldots, p$ contains a path from the V_2^i seed to the Z seed but does not contain any paths from the V_2^i seed to any other V_2^j seed, $j \neq i$.
  - Subtree $p + 1$ (which may be empty) connects the Z seed to other vertices, but does not contain any paths from Z to any of the V_2^i seeds.

  Once the structure has been made, we iteratively shrink vertices of degree one (different from the V_2^i and Z seeds) into their unique adjacent vertex to obtain a (not necessarily regular) PB configuration.

- **Case 2:** $b = 0$. Here, $p \geq 3$, it is possible for A to be empty and we do not have a 'seed' for Z. We build *two* structures, one in which Z is forced to be empty and one in which Z is forced to non-empty.

  In the first case, we build a *forest* in G_s which spans all vertices in V_s apart from those representing A and the V_1^i. It must consist of p vertex-disjoint trees, such that each of the V_2^i seeds is contained in exactly 1 tree. Once the forest is made, we pick longest paths going out from each of the V_2^i seeds to get the p necessary paths and shrink vertices of degree 1 as before.

  In the second case, we build a tree in G_s which spans all vertices in V_s apart from those representing A and the V_1^i. Now, an edge is permitted to create a path connecting two distinct V_2^i seeds. Once this happens, one of the vertices on the path (different from the V_2^i seeds) must now be a Z seed. The remainder of the tree is built with this in mind: The addition of an edge forming a path connecting one of the possible Z seeds to a third V_2^i seed is not permitted unless it is incident on a potential Z seed (which is then chosen as the actual Z seed). If this never happens (because the shrunk graph is not sufficiently connected), the tree is abandoned. Otherwise, we proceed as in case 1 now that we have our Z seed.

At this stage, we will generally have a non-regular PB configuration (or possibly two). Moreover, the structure of the configuration in the vicinity of Z is likely to be 'messy'. We deal with both of these problems simultaneously: We choose n according to the length of the shortest of the p paths in the configuration and shrink any paths which are longer than this by merging Z with adjacent V_j^i.

Now that we have an n-PB inequality, we check if is violated. If not, we may still be able to obtain a violated inequality by 'removing handles' (analogous to shrinking a K-C configuration). It is easy to see from the inequality (6) that the slack of the n-PB inequality will be decreased by removing any handle H_i such that $x^*(\delta(H_i)) > p + 1$ and merging adjacent V_j^i accordingly. This is done iteratively, as long as $n \geq 2$ remains.

3.7 Honeycomb separation

As for the K-C inequalities, we first try to separate honeycomb inequalities in which a single R-connected component is partitioned into more than 2 parts. The effect of these honeycomb inequalities is to separate solutions x^* where some R-even vertices $u_1, u_2, \ldots, u_L$ (or sets with an even number of R-odd vertices) belonging to an R-set C_i satisfy $x^*(\delta(u_j)) \cong 1$ and $x^*(E(\{u_j\} : C_i \setminus \{u_j\})) \cong 0$, $j = 1, 2, \ldots, L$.

The idea of our algorithm is, given an R-connected component, to find the x^*-external vertices, say $u_1, u_2, \ldots, u_t$, and, then, assign the remaining vertices to some of the previous ones in order to define vertex sets $V_1, V_2, \ldots, V_t$ in such a way that $x^*(E(V_i : V_j))$ is as small as possible, for $1 \leq i < j \leq t$. The procedure is as follows:

Phase I: Define seeds for $V_1, V_2, \ldots, V_L$.
Given an R-set C_i, we assign to each x^*-external vertex u_j a label k corresponding to the R-set C^k with $x^*(\{u_j\} : C^k)$, $k \neq i$ maximum. To each of the remaining vertices in C_i, a different negative label is assigned. Starting with the edge (u, v) in $E(C_i)$ with largest x^*-weight, let l_{min} (l_{max}) be the smaller (larger) label of that of u and v. We assign the label l_{max} to all vertices in C_i having label l_{min}. This procedure is repeated until all vertices

have positive label. Vertices with the same label define a partition $V_1, V_2, \ldots, V_t$ of the set of vertices of C_i. If the number of R-odd vertices in each $V_j, j = 1, \ldots, t$ is even, we are done. Otherwise, all the sets V_j with an odd number of R-odd vertices are joined, forming a single set. Hence, we have defined a partition $V_1, V_2, \ldots, V_L$. If $L \geq 3$, these sets are suitable to be considered as part of a honeycomb configuration. If $L = 2$, these sets are suitable to be considered as V_0 and V_K for a K-C configuration. Since this way of defining V_0 and V_K is different from that of Phase I in the above K-C separation algorithm, it is worthwhile to look for a violated K-C inequality. We then apply Phases II and III of K-C separation. Otherwise ($L = 1$), we study another R-connected component.

Phase II: Define the vertex sets $W_1, W_2, \ldots, W_{K-1}$ and the tree.
After (seeds for) $V_1, V_2, \ldots, V_L$ have been defined, we consider the graph obtained by shrinking each $V_1, V_2, \ldots, V_L$ and each of the remaining R-sets into single vertices. As in the separation of K-C and n-PB inequalities, we try to compute a spanning tree with large x^* weight in this shrunk graph, by iteratively adding the edge of maximum weight not forming a cycle (and not crossing between the V_i). Initially we forbid adding edges of zero weight. If we obtain a tree without having to add an edge of zero weight, we then shrink (iteratively) each vertex with degree one on the tree (different from $V_1, V_2, \ldots, V_L$) into its (unique) adjacent vertex, thus obtaining the vertex sets $W_1, W_2, \ldots, W_M$ and the tree defining the honeycomb configuration. If we do not, then before adding edges of zero weight, we check each one of the connected components in the forest to see whether it defines a honeycomb configuration by itself. Only if this also fails do we add to the forest the edges of zero weight needed to get a spanning tree. If the degree of any vertex V_i on the tree is more than two, we reject that honeycomb configuration and consider another R-connected component. Otherwise, for each vertex V_i incident with two vertices, W_j and W_k, on the tree, we replace one of the edges (V_i, W_j) or (V_i, W_k) (that with largest x^*-weight) by edge (W_j, W_k) on the tree. Then, every vertex V_i has degree one in the configuration tree.

If the above procedure has been successful, we now have a honeycomb configuration defined by $V_1, V_2, \ldots, V_L$ and $W_1, W_2, \ldots, W_{K-1}$ and its corresponding inequality $F(x) \geq 2(K-1)$.

Phase III: Check the Honeycomb inequality.
For a given vector x^*, in order to compute $F(x^*)$, we define a vector with a record $T(e)$ for each edge in the configuration tree. For each edge $e \in E$, we add x^*_e to the records corresponding to the edges in the (unique) path on the tree joining the two 'shrunk' vertices containing the two end-vertices of edge e. Then, we can compute easily

$$F(x^*) = \sum_{e \in T} T(e) - 2 \sum_{1<i<j<L} x^*(V_i : V_j) \tag{8}$$

Given a honeycomb configuration, if we shrink any pair of sets W_i, W_j adjacent in the tree into a single one, we will obtain a new 'smaller' honeycomb inequality. Furthermore, when we shrink W_i, W_j, the obtained inequality will be of the form $F'(x) \geq 2(K-2)$, where $F'(x^*) = F(x^*) - T(i,j)$. Hence, shrinking sets W_i, W_j with $T((W_i, W_j)) > 2$ leads to honeycomb inequalities with greater violation. As for the K-C inequalities, if a given honeycomb inequality is not violated, we check if we will obtain a new violated honeycomb inequality by shrinking pairs of adjacent vertices on the tree and if so we shrink them accordingly. We do not shrink sets of the form V_i, W_j, since doing this could lead to a honeycomb configuration in which a vertex of type V_j has degree more than one in the tree, a situation which was forbidden in Subsection 2.6 to make the computation of coefficients tractable.

As for the separation of K-C inequalities (Subsection 3.4), the above procedure can also be extended to search for violated honeycomb inequalities in which the required edges crossing between V_i and V_j belong to more than one R-connected component.

4 The Overall Algorithm

In this section we will describe how the separation algorithms are integrated to form the overall cutting-plane algorithm.

4.1 Initial Relaxation

The first stage in the algorithm is to construct an initial LP relaxation to be input to the LP solver. We have found that a good choice is to include one connectivity inequality for each R-component and one R-odd cut inequality for each R-odd vertex (together with the non-negativity inequalities, which are handled implicitly by the LP-solver). We also found it useful to include the upper bounds $x_e \leq 2$ for each $e \in E$. (It is easy to show that these upper bounds hold in any optimal solution).

We also want to mention that, for some problems, it is possible to fix some variables at zero. If, for three vertices i, j, k, the cost of edge $\{i, j\}$ is equal to the sum of the costs of edges $\{i, k\}$ and $\{k, j\}$, then edge $\{i, j\}$ is redundant and its associated variable can be removed from the problem. However, this criterion only applied to a few of our test instances (see Section 5).

4.2 Inequalities Added in Each Iteration

In each iteration of the cutting-plane algorithm, the exact and heuristic separation algorithms are invoked in a specific order and a number of violated inequalities are added to the LP relaxation, which is then resolved using the dual simplex method. We have experimented with various priority orderings for the separation algorithms. The following ordering works very well for problems in which $E_R \neq \emptyset$:

1. R-odd cut and connectivity separation heuristics with $\epsilon = 0$.
2. If no violated inequality was found in step 1, then repeat with $\epsilon = 0.25$.
3. If no violated inequality was found in step 2, then repeat with $\epsilon = 0.5$.
4. Exact connectivity separation if the heuristics failed.
5. Exact R-odd cut separation if the heuristics failed.
6. If the number of violated inequalities detected so far is > 10, stop.
7. K-C separation heuristic with $\epsilon = 0$.
8. If no violated inequality was found in step 7, then repeat with $\epsilon = 0.25$.
9. If no violated inequality was found in step 8, then repeat with $\epsilon = 0.5$.
10. If the number of violated inequalities detected so far is > 15, stop. Else do honeycomb separation heuristic (that can also find K-Cs).
11. If the number of violated inequalities detected so far is > 15, stop. Else do n-PB heuristic with $\epsilon_1 = 0, 0.25, 0.5, 0.75$ and 1 and $\epsilon_2 = \min\{0.25, \epsilon_1\}$.
12. If the number of violated inequalities detected so far is > 5, stop. Else do heuristics for K-Cs and honeycombs with iterative merging of adjacent R-components.
13. If no violated inequality has been found so far, then do 2-PB heuristic (which is exact

for simple 2-PBs).

When $E_R = \emptyset$ (i.e., when the GRP instance is a SGTSP instance), it is better to use the following strategy:

1. Connectivity separation heuristics with $\epsilon = 0$.
2. If no violated inequality was found in step 1, then repeat with $\epsilon = 0.25$.
3. If no violated inequality was found in step 2, then repeat with $\epsilon = 0.5$.
4. If the number of violated inequalities detected so far is > 15, stop. Else do n-PB heuristic with $\epsilon_1 = 0, 0.25, 0.5, 0.75$ and 1 and $\epsilon_2 = \min\{0.25, \epsilon_1\}$.
5. If no violated inequalities have been detected so far (or, every fifth iteration, no connectivity inequalities have been found so far), do exact connectivity separation algorithm.
6. If no violated inequality has been found so far, then do 2-PB heuristic.

4.3 Constraint Management and the Cut Pool

When a large number of cuts have been added, the LPs can become rather large, causing speed and memory problems. To alleviate this problem, it is helpful to occasionally 'purge' constraints from the LP formulation. We found that a good strategy was to remove, every 10 iterations, any constraints which have a slack of 0.01 or more.

However, it can happen that a constraint which was removed at one time can become violated in a subsequent iteration. Because of the heuristic nature of some of the separation algorithms, this could lead to a useful constraint being lost forever. To avoid this difficulty, we use the idea of a *cut pool* [27]. Every time a violated inequality is found, it is stored in the cut pool. Then, whenever the LP is purged, the cut pool is checked to see if it contains any violated constraints. If so, these are added to the LP and the LP is then re-solved.

We also search through the cut pool whenever the separation algorithms fail to find a violated inequality.

4.4 Additional Phase

If no more violated inequalities can be found and the solution vector x^* still does not represent an optimal tour, we do some more things.

First, we use a recent result in [11]. Consider the graph obtained by shrinking each R-set of G into a single vertex, but *without* merging parallel edges. Construct a minimum cost spanning tree T on this graph (using the c_e as costs). Then there is an optimal solution to the GRP such that the following conditions hold:

- $x_e \leq 2$ for all $e \in T$,
- $x_e \leq 1$ for all $e \notin T$.

These upper bounds dominate the ones mentioned in Subsection 4.1. We insert these upper bounds into the LP relaxation and then call the separation procedures again. (The reason that we do not put these upper bounds in from the very start is that, on some instances, their presence can create spurious 'odd' vertices in the LP relaxation.)

If this still does not yield the optimal tour and the lower bound is not an integer value, we append an extra inequality which enforces integrality of the objective and then call the separation procedures again. (For example, if the lower bound was 100.23, then we would add the constraint $\sum_{e \in E} c_e x_e \geq 101$.)

If, after all this, the LP relaxation is still not integral, we invoke branch-and-bound. To avoid the solution of very large integer programs (IPs), we first delete all constraints with a slack of 0.01 or more. We have found that the resulting IP is usually solved quickly and often leads to the optimal solution of the GRP. However, it is possible that the IP solution might not be feasible for the GRP, since the associated 'tour' could fail to be connected, Eulerian, or both. In such a case, the procedure terminates with a (very tight) lower bound, but no feasible GRP solution.

5 Computational Experiments

The algorithm has been coded in the *C* language and run on a SUN Sparc 20 workstation, using the *CPLEX* library to solve the LP problems. (Note that the SUN machine is rather slow. Its speed is comparable to a 66 MhZ Pentium). The algorithm has been tested on RPP and GTSP instances, as well as 'genuine' GRP instances. A description of the test instances follows:

- 24 RPP instances, denoted by I1 to I24, from [4] and two more RPP instances, denoted by IAA and IAB, from [5] obtained by randomly selecting some edges in the real-world based graph, with 116 vertices and 174 edges, of the city of Albaida (Valencia) (Table 1).
- 92 RPP instances from [15]. In these problems, the vertices lie on the Euclidean plane and edge costs are proportional to Euclidean distances. The set E is formed according to one of three procedures and then the set E_R is chosen randomly. In the first procedure (used for 20 instances, Table 2), E is made of edges with short length in the plane. In the second procedure, (used for 36 instances, Table 3), E forms a unit grid graph. In the third procedure (also used for 36 instances, Table 4), G is a graph in which all vertices have degree 4, made from two edge-disjoint Hamiltonian cycles found by a nearest neighbour heuristic. These instances have $16 \leq |V| \leq 100$ and $24 \leq |E| \leq 200$.
- 10 GRP instances generated from the Albaida graph by visually selecting the required edges. We select, for example, edges corresponding to streets oriented from north to south, from east to west, forming cycles, etc. (Table 5).
- 15 GRP instances randomly generated from the Albaida graph, by defining each edge as required with probability P and considering all the vertices of the graph as required. Five graphs were generated for each value of $P = 0.7, 0.5$ and 0.3, named alb-7-1 to alb-3-5 (Table 6).
- 15 GRP instances, named mad-7-1 to mad-3-5, randomly generated from the real-world based graph of the city of Madrigueras (Albacete) — with 196 vertices and 316 edges — in the same way as for Albaida (Table 7).
- 7 GTSP instances with between 150 and 225 vertices and between 296 and 433 edges (Table 8). These were formed by taking planar Euclidean TSP instances from TSPLIB [28] and making the associated graphs sparse by deleting all edges apart from
  - The edges in an optimal TSP tour
  - The edges connecting each vertex to its three nearest neighbours.

For these instances, the cost of the optimal solution was known beforehand. Also, some of the edges in these instances could be eliminated using the criterion given in Subsection 4.1. However, we only did this for the two largest instances (rat195g and ts225g).

In the following tables, LB0 shows the lower bound values corresponding to the initial relaxations, while LB shows the lower bound values obtained at the end of the cutting-plane procedure. For the cases where branch and bound was used, the solution cost is given in the B&B column. Total running times have also been included. For each instance in Tables 1 and 5 to 8, the number of identified violated connectivity, R-odd cut, K-C, honeycomb, K-C and honeycombs with merging of adjacent R-sets, n-PB and simple 2-PB constraints are presented under headings conn, odd, K-C, HC, KC2, HC2, nPB and SPB, respectively.

In all tables, a '*' on a LB or B&B entry means that not only the optimal value but also an optimal GRP solution was obtained using the cutting-plane algorithm alone, or the cutting-plane algorithm plus B&B, respectively. A '*' on an UB entry means that this upper bound is optimal.

In Tables 2 to 4 the entries corresponding to the different types of constraints and to the running time represent the accumulated amount for all the instances in each group. Moreover, the number of instances in each group solved to optimality with cutting-planes alone and the number of instances that were solved to optimality after B&B are shown.

Finally, the upper bounds shown in Table 5 for instances grp4 and grp10 were obtained by Alain Hertz (private communication) using the methods in [15]. Instances grp4 and grp10 were built to be difficult, but it is our belief that both upper bounds should be optimal or very near optimal, as the sophisticated heuristic procedures of [15] produced optimal solutions for all the instances reported in Tables 1 to 4 and for the 8 other instances of Table 5.

It will be seen that 116 out of 118 RPP instances were solved to optimality by cutting planes alone and the other 2 were solved after invoking branch and bound. The maximum percentage gap between the LP lower bound value and the optimal cost was 0.26% for these instances. Thus, the algorithm works extremely well for RPP instances.

When it comes to 'genuine' GRP instances, it will be seen that 28 out of 40 instances were solved to optimality by cutting planes alone and a further 6 were solved by branch and bound. The maximum percentage gaps between the LP lower bound and the optimal cost or upper bound were 1.81% for instance grp10 in Table 5, 0.5% for instance alb-3-1 in Table 6 and 0.37% for instance mad-5-5 in Table 7.

Finally, 3 out of the 7 GTSP instances, 6 if we resorted to branch and bound, were solved to optimality. The worst performance was obtained for the instance ts225g. For this instance, the percentage gap was 1.62%, and the branch-and-bound algorithm failed to find the optimal integer solution after 20,000 nodes had been explored. (Note, however, that the corresponding TSPLIB instance, ts225, was deliberately constructed to be difficult for cutting plane algorithms). It will also be noticed that the running times were somewhat larger for the GTSP instances than for the RPP and GRP instances. Even so, they are acceptable given that our machine is at least 8 times slower than a modern PC.

6 Conclusions

The computational results given in the previous section show that our cutting plane algorithm is capable of solving many GRP instances of realistic size to proven optimality. This provides further evidence, if evidence were needed, that the study of integer polyhedra is important from a practical as well as a theoretical point of view.

Of course, there is still room for improvement in our algorithm. One obvious improvement would be the addition of better separation algorithms. In this context, 'better' could simply mean faster, or it could mean generating more inequalities or inequalities of greater generality. Another improvement would be to embed the separation algorithms within a full branch-and-cut scheme (see [27]).

Finally, we would also like to mention that the separation procedures presented in this paper could be useful for more complex routing problems, perhaps containing multiple vehicles, directed arcs or other complications encountered in practice (see [9] for a survey).

Acknowledgements: The authors want to thank Alain Hertz for providing us his instances and upper bounds.

References

[1] D. Applegate, R.E. Bixby, V. Chvátal & W. Cook (1995) Finding cuts in the TSP. *DIMACS Technical Report TR 95-05*, Rutgers University, New Brunswick, NJ, USA.

[2] P. Augerat, J.M. Belenguer, E. Benavent, A. Corberán, D. Naddef & G. Rinaldi (1995). Computational results with a branch-and-cut code for the capacitated vehicle routing problem. Research report RR949-M, ARTEMIS-IMAG, France.

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

[4] N. Christofides, V. Campos, A. Corberán & E. Mota (1981) An algorithm for the rural postman problem. *Imperial College Report IC.OR.81.5.* London.

[5] A. Corberán & J.M. Sanchis (1994) A polyhedral approach to the rural postman problem. *Eur. J. Opl Res.*, *79*, 95-114.

[6] A. Corberán & J.M. Sanchis (1998) The general routing problem polyhedron: facets from the RPP and GTSP polyhedra. *Eur. J. Opl Res.*, *108*, 538-550.

[7] G. Cornuéjols, J. Fonlupt & D. Naddef (1985) The travelling salesman on a graph and some related integer polyhedra. *Math. Prog.*, *33*, 1-27.

[8] J. Edmonds & E.L. Johnson (1973) Matchings, Euler tours and the Chinese postman problem. *Math. Prog.*, *5*, 88-124.

[9] H.A. Eiselt, M. Gendreau & G. Laporte (1995) Arc-routing problems, part 2: the rural postman problem. *Oper. Res.*, *43*, 399-414.

[10] B. Fleischmann (1985) A cutting-plane procedure for the travelling salesman problem on a road network. *Eur. J. Opl Res.*, *21*, 307-317.

[11] G. Ghiani & G. Laporte (1997) A branch-and-cut algorithm for the undirected rural postman problem. *Working paper*, Centre for Research on Transportation, University of Montréal.

[12] M. Grötschel & O. Holland (1985) Solving matching problems with linear programming. *Math. Prog.*, *33*, 243-259.

[13] M. Grötschel & M.W. Padberg (1979) On the symmetric travelling salesman problem I: inequalities. *Math. Prog.*, *16*, 265-280.

[14] M. Grötschel & Z. Win (1992) A cutting plane algorithm for the windy postman problem. Math. Prog., *55*, 339-358.

[15] A. Hertz, G. Laporte & P. Nanchen (1998) Improvement procedures for the undirected rural postman problem. *Working paper*, Centre for Research on Transportation, University of Montréal.

[16] M. Jünger, G. Reinelt & G. Rinaldi (1995) The travelling salesman problem. In Ball et al. (Eds.) *Network Models*. Handbooks on Operations Research and Management Science Vol. 7. Amsterdam: Elsevier.

[17] M. Jünger, G. Reinelt & G. Rinaldi (1997) The travelling salesman problem. In Dell'Amico et al. (Eds.) *Annotated Bibliographies in Combinatorial Optimization.* New York: Wiley.

[18] J.K. Lenstra & A.H.G. Rinnooy-Kan (1976) On general routing problems. *Networks*, *6*, 273-280.

[19] A.N. Letchford (1997a) New inequalities for the general routing problem. *Eur. J. Opl Res.*, *96*, 317-322.

[20] A.N. Letchford (1997b) The general routing polyhedron: a unifying framework. To appear in *Eur. J. Opl Res.*, *Vol. 112* (1999).

[21] H. Nagamochi, T. Ono & T. Ibaraki (1994) Implementing an efficient minimum capacity cut algorithm. *Math. Prog.*, *67*, 325-341.

[22] G.L. Nemhauser & L.A. Wolsey (1988) *Integer and Combinatorial Optimization.* New York: Wiley.

[23] Y. Nobert & J.-C. Picard (1996) An optimal algorithm for the mixed Chinese postman problem. *Networks*, *27*, 95-108.

[24] C.S. Orloff (1974) A fundamental problem in vehicle routing. *Networks*, *4*, 35-64.

[25] M.W. Padberg & M.R. Rao (1982) Odd minimum cut-sets and b-matchings. *Math. Oper. Res.*, *7*, 67-80.

[26] M.W. Padberg & G. Rinaldi (1990) Facet identification for the symmetric travelling salesman polytope. *Math. Prog.*, *47*, 219-257.

[27] M.W. Padberg & G. Rinaldi (1991) A branch-and-cut algorithm for the resolution of large-scale symmetric travelling salesman problems. *SIAM Rev.*, *33*, 60-100.

[28] G. Reinelt (1991) TSPLIB - a travelling salesman problem library. *ORSA J. on Comput.*, *3*, 376-384.

[29] R.E. Tarjan (1972) Depth first search and linear graph algorithms. *SIAM J. on Comp.*, *1*, 146-60.

				constraints								
	$\|V\|$	R-sets	LB0	conn.	odd	K-C	KC2	HC	HC2	nPB	LB	time (scs.)
I01	11	4	31	5	10	-	-	-	-	-	51*	0.03
I02	14	4	62	4	10	4	-	-	-	-	72*	0.04
I03	28	4	24	5	16	9	4	-	-	-	29*	0.10
I04	17	3	24	3	14	7	-	-	-	-	29*	0.10
I05	20	5	49	5	14	14	9	-	-	-	55*	0.17
I06	24	7	27	8	24	13	6	-	-	-	32*	0.23
I07	23	3	35	3	18	-	-	-	-	-	37*	0.08
I08	17	2	26.5	2	16	-	-	-	-	-	29*	0.09
I09	14	3	18	3	12	1	-	-	-	-	26*	0.06
I10	12	4	27	5	10	8	-	-	-	-	35*	0.10
I11	9	3	9	3	6	-	-	-	-	-	9 *	0.08
I12	7	3	6	3	6	2	-	-	-	-	6 *	0.02
I13	7	3	23	3	6	-	-	-	-	-	23*	0.01
I14	28	6	43	9	28	18	6	-	-	-	57*	0.32
I15	26	8	207	16	30	2	1	-	-	-	261*	0.23
I16	31	7	53	8	25	14	3	6	-	2	64*	0.64
I17	19	5	39	7	16	9	6	2	1	-	49*	0.17
I18	23	8	54	12	26	-	-	1	1	-	85*	0.17
I19	33	7	112	7	30	3	-	-	-	-	116*	0.12
I20	50	7	80	12	34	26	8	3	2	-	116*	0.54
I21	49	6	52	9	40	41	19	8	3	-	78 *	1.25
I22	50	6	103	7	30	3	-	1	-	-	122*	0.31
I23	50	6	78	10	45	51	32	2	2	2	95 *	1.18
I24	41	7	96	8	33	28	16	6	5	2	113*	1.17
IAA	102	10	2896	11	114	105	90	15	15	3	3304*	6.98
IAB	90	11	1728	17	87	11	4	4	2	-	2826*	2.42

Table 1: Computational results for the Christofides et al (1981) and Corberán & Sanchis (1994) instances.

n	# of inst.	R-sets average	TOTAL constraints for all 5 instances							# of LB opt.	# of B&B opt.	Total time (scs.) for all 5 instances
			conn.	odd	K-C	KC2	HC	HC2	nPB			
20	5	3.4	18	44	2	-	-	-	-	5	-	0.20
30	5	4.8	30	80	24	13	-	-	-	5	-	0.56
40	5	7.0	46	107	31	17	2	1	2	5	-	1.18
50	5	8.2	59	130	13	10	4	2	4	5	-	1.60

Table 2: Computational results for Class 1 RPP random instances from Hertz et al. (1998).

n	Prob.	# of inst.	R-sets average	TOTAL constraints for all 3 instances							# of LB opt.	# of B&B opt.	Total time (scs.) for all 3 instances
				conn.	odd	K-C	KC2	HC	HC2	nPB			
	0.3	3	4.0	13	26	-	-	-	-	-	3	-	0.11
16	0.5	3	4.3	13	28	2	2	-	-	-	3	-	0.11
	0.7	3	4.0	14	30	19	6	-	-	-	3	-	0.21
	0.3	3	8.3	33	72	45	28	-	-	1	3	-	1.0
36	0.5	3	6.6	24	96	46	23	2	2	-	3	-	1.22
	0.7	3	5.6	22	91	64	34	9	6	-	3	-	1.28
	0.3	3	12.0	57	137	51	22	3	2	4	3	-	2.75
64	0.5	3	9.0	35	226	102	43	14	9	1	3	-	9.19
	0.7	3	5.0	21	151	80	26	-	-	-	3	-	2.97
	0.3	3	19.6	86	243	97	43	28	12	9	3	-	16.6
100	0.5	3	14.0	72	977	571	312	154	81	3	2	1	109
	0.7	3	6.3	35	528	281	107	66	24	-	3	-	42.4

Table 3: Computational results for Class 2 RPP random instances from Hertz et al. (1998).

n	Prob.	# of inst.	R-sets average	TOTAL constraints for all 3 instances							# of LB opt.	# of B&B opt.	Total time (scs.) for all 3 instances
				conn.	odd	K-C	KC2	HC	HC2	nPB			
	0.3	30	3.0	10	24	6	2	-	-	-	3	-	0.09
16	0.5	3	3.6	13	26	4	-	-	-	-	3	-	0.19
	0.7	3	4.6	16	34	8	2	-	-	-	3	-	0.22
	0.3	3	8.3	30	72	20	12	-	-	-	3	-	0.59
36	0.5	3	7.3	25	74	29	19	-	-	-	3	-	0.79
	0.7	3	5.3	20	106	31	8	-	-	-	3	-	0.96
	0.3	3	12.3	45	122	49	34	3	2	1	3	-	2.04
64	0.5	3	10.3	43	273	106	57	13	6	6	3	-	8.61
	0.7	3	6.6	23	192	69	40	7	4	3	3	-	3.46
	0.3	3	20.3	93	325	93	57	5	4	4	3	-	21.8
100	0.5	3	13.0	61	275	82	26	14	4	5	3	-	8.08
	0.7	3	9.0	38	348	151	57	32	4	-	2	1	23.6

Table 4: Computational results for Class 3 RPP random instances from Hertz et al. (1998).

	R-sets	LB0	constraints							LB	B&B	UB	time (scs.)	# of iter.
			conn.	odd	K-C	KC2	HC	HC2	nPB					
grp1	56	4183	84	127	53	16	8	3	9	5625*			42.1	14
grp2	52	3748	87	194	128	96	22	9	6	5569*			506	36
grp3	55	3937	90	125	31	16	4	4	1	5688*			65.4	13
grp4	48	2576	114	45	311	26	350	49	-	5119	5158	5186	1156	67
grp5	45	3620	68	115	26	16	12	6	2	5191*			119	13
grp6	10	2568	13	92	-	-	-	-	-	3442*			1.3	4
grp7	64	4525	103	113	52	27	18	4	2	6618*			110	14
grp8	47	5044	85	132	44	16	17	2	3	6814*			31.5	17
grp9	60	3329	93	92	17	12	15	8	3	4506*			207	15
grp10	48	2480	121	-	404	66	429	82	-	5029	5070	5122	2011	85

Table 5: Computational results for 10 visually generated instances from the Albaida graph (116 vertices and 174 edges).

	R-sets	LB0	constraints								LB	B&B	UB	time (scs.)	# of iter.
			conn.	odd	K-C	KC2	HC	HC2	nPB	SPB					
7-1	18	2789	24	242	100	84	3	1	-	2	4374*			106	63
7-2	10	2538	10	122	3	1	2	-	-	-	3562*			3.0	14
7-3	15	2480	20	178	242	225	36	31	2	-	3559*			33.0	43
7-4	10	2456	11	190	-	-	-	-	-	-	3582*			20.2	49
7-5	13	2714	15	155	22	21	3	3	-	-	4077*			30.9	34
5-1	33	3374	45	144	125	93	19	12	8	-	4785*			84.8	24
5-2	30	3219	42	124	10	10	4	1	-	-	4641*			26.0	9
5-3	28	3432	38	106	15	5	8	-	-	-	4322*			13.0	11
5-4	33	3140	46	107	43	24	10	-	2	3	4714	4719*		102	23
5-5	30	3400	41	106	21	-	1	-	-	-	4332*			6.2	8
3-1	66	4042	115	160	56	25	30	18	4	2	5703	5712	5732*	277	29
3-2	71	4666	122	122	57	34	16	6	15	-	6699	6716*		194	25
3-3	72	4490	133	99	22	12	4	3	10	-	6189	6189	6201*	192	20
3-4	67	4165	120	135	84	25	11	5	8	-	5926*			139	25
3-5	62	4395	97	107	26	6	24	15	4	-	5856*			147	13

Table 6: Computational results for 15 randomly generated instances from the Albaida graph (116 vertices and 174 edges).

	R-sets	LB0	constraints								LB	B&B	UB	time (scs.)	# of iter.
			conn.	odd	K-C	KC2	HC	HC2	nPB	SPB					
7-1	15	4325	19	160	35	19	8	4	-	-	5260*			11.0	10
7-2	6	4520	6	172	-	-	-	-	-	-	4930*			7.8	11
7-3	11	4322.5	12	164	8	6	-	-	-	-	5135*			9.8	10
7-4	7	4047.5	8	162	-	-	3	-	-	-	4775*			9.4	14
7-5	8	4617.5	8	192	-	-	-	-	-	-	5375*			9.0	17
5-1	41	6010	51	206	100	92	14	3	3	-	6920*			450	25
5-2	47	5715	57	172	17	-	8	-	-	-	6560*			20.5	7
5-3	53	6160	73	222	104	73	25	14	5	-	6949	6955*		1351	29
5-4	53	5857	82	228	180	64	79	25	12	-	6915*			499	39
5-5	53	5900	70	204	39	16	10	-	5	-	6765	6780	6790*	174	18
3-1	111	7680	180	338	172	47	25	6	12	-	8672	8680*		2515	44
3-2	88	7037.5	127	237	50	14	19	-	20	2	8155*			491	21
3-3	95	7295	163	702	122	43	41	7	9	-	8526	8555	8555*	3115	64
3-4	95	7355	153	395	134	55	45	15	21	2	8665	8680*		2375	48
3-5	103	7645	161	250	97	19	26	-	11	2	8745	8755*		1871	32

Table 7: Computational results for 15 randomly generated instances from the Madrigueras graph (196 vertices and 316 edges).

	LB0	constraints			LB	B&B	UB	time (scs.)	# of iter.
		conn.	nPB	SPB					
kroA150g	21604.5	282	24	-	26489	26524*		1045	23
kroA200g	23491.5	361	23	-	29368*			1142	15
kroB150g	20761.5	264	21	-	26130*			332	15
kroB200g	23888	342	8	-	29435	29437*		775	13
pr152g	43232	360	-	-	73682*			74	11
rat195g	2097	369	48	12	2317	2323*		15773	73
ts225g	115605	617	617	-	124591	125052	126643*	17976	265

Table 8: Computational results for 7 GTSP instances generated from TSP instances in TSPLIB.