Mathematical Programming 55 (1992) 339-358
North-Holland

A cutting plane algorithm for the windy postman problem

M. Grötschel

Konrad-Zuse-Zentrum für Informationstechnik, W-1000 Berlin 31, Germany, and Technische Universität Berlin, W-1000 Berlin 12, Germany

Zaw Win

Department of Mathematics, University of Rangoon, Rangoon, Burma

Received 14 September 1989
Revised manuscript received 12 October 1990

In this paper we describe a cutting plane algorithm for the (NP-hard) windy postman problem. This algorithm can also be applied to the mixed, directed, and undirected postman problems. It is based on a partial linear description of the windy postman polyhedron and on the simplex method. The partial linear description (together with cutting plane recognition strategies) provides new cutting planes and hence generates better and better linear programming relaxations of the windy postman polyhedron, and the simplex method solves these linear programs. We have investigated the performance of our algorithm with several test problems defined on graphs of up to 264 nodes. For most of these problems, we obtained optimal solutions in reasonable computation times.

1. Introduction

The following is a combinatorial optimization problem which arises in the postal service.

> "A postman wants to deliver the mail in his town. He starts at his office, delivers the mail along all streets and then goes back to the office. The question is to design a tour of minimum cost for him."

In the literature, the above problem and similar ones are known as *postman problems*. In the mathematical formulations of these problems, real world transportation networks are represented by (undirected) graphs, digraphs or mixed graphs, and the corresponding postman problems are called *undirected*, *directed*, or *mixed* respectively. (The undirected postman problem is also known as the *Chinese postman problem* (short: CPP) since it was proposed by the Chinese mathematician M. Guan, see Guan (1962).) In many real world problems, the cost of traversing a street in one direction is the same as that of traversing it in the opposite direction. But, for

some problems, this is not the case. These are called *windy postman problems* (short: WPP). (On a windy day, the cost of traversing a street with the wind will be different from that of traversing it against the wind.) Although the above types of routing problems are termed as postman problems, they arise also in many other areas, for instance, in waste collection, street cleaning, snow removal, road inspection, and school bus routing etc.

In this paper we present a cutting plane algorithm for the WPP. The algorithm is based on a partial linear description of the windy postman polyhedron described in Win (1987) and Grötschel and Win (1992) and on the simplex method. As the initial linear program (short: LP) we solve the LP relaxation of a canonical integer linear programming formulation of the WPP. At each iteration, some inequalities that are not binding at the current fractional LP solution are deleted from the current LP, inequalities (that are in our partial linear description and) violated by the current fractional LP solution are recognized and added to the current LP, and the updated LP is reoptimized. The algorithm stops when the current LP solution is integral, or if it is fractional and violates no inequality of our partial linear description. In the first case, we get an optimal windy postman tour while, in the other, we use the fractional LP solution to construct an approximate windy postman tour heuristically.

Thus our algorithm is not exact, i.e., it does not guarantee termination with an optimal solution. Our computational results of Section 5, however, show that the algorithm generates optimal tours in most cases, and in the others, approximate tours are produced that are almost optimal. Moreover, one can transform our algorithm into an exact procedure by supplementing an enumeration phase like branch and bound. Since the undirected, directed and mixed postman problems can be viewed as special cases of the WPP, one can apply our algorithm also to such instances.

The paper is organized as follows. In Section 2 we describe some notation, definitions and statements of various postman problems. In Section 3 we summarize some polyhedral results on which our cutting plane algorithm is based. Then, in Section 4, we give a detailed description of the algorithm. The practical performance of our algorithm is investigated by running it on several test problems. These computational results are reported in Section 5.

2. Notation, definitions, and statements of the problems

Most of the graph and polyhedral theory terms used in this paper are standard — see, for example, Bondy and Murty (1976) and Bachem and Grötschel (1982). We thus introduce only those that are used frequently or are nonstandard.

We denote (undirected) *graphs* by $G = (V, E)$ where $V \neq \emptyset$ is the set of *nodes* and E the set of *edges*. *Loops* are irrelevant for our purposes, so we assume throughout that all graphs are loopless. We allow *parallel edges* since, for instance, parallel

roads sometimes occur in road networks. An edge with endnodes i and j will be denoted by ij although this is slightly imprecise in case there are more edges than one *incident with* i and j. We make sure that this notation does not lead to confusion. For a node $i \in V$, we denote the set of edges incident with i by $\delta(i)$; its cardinality is called the *degree* of i. An *even* resp. *odd node* is a node with even resp. odd degree. *Digraphs* are denoted by $D = (V, A)$ where $V \neq \emptyset$ is the set of nodes and A the set of *arcs*. As above, loops are neglected and parallel arcs are allowed. We denote an arc $a \in A$ *incident from* i *to* j by $a = (i, j)$. We denote *mixed graphs* by $M = (V, E, A)$ where $V \neq \emptyset$ is the set of nodes, E the set of edges and A the set of arcs.

Let $G = (V, E)$ be a graph and $u, v \in V$. A sequence $\omega = \langle i_0 i_1, i_1 i_2, \ldots, i_{k-1} i_k \rangle$ of edges of E with $i_0 = u$, $i_k = v$ is called a *walk* between u and v. ω is *open* if $u \neq v$ and *closed* otherwise. ω is a *path* if $i_s \neq i_t$ for $0 \leq s < t \leq k$. If ω is a path and $i_k i_0 \in E$, then the sequence $\langle i_0 i_1, i_1 i_2, \ldots, i_{k-1} i_k, i_k i_0 \rangle$ is a *cycle*. If a walk exists between any pair of nodes of V, then G is said to be *connected*. A subgraph of a graph G that is connected and that is maximal with respect to this property is called a (connected) *component* of G. Any edge e of G such that the graph H obtained from G by removing e has more components than G is called a *bridge* of G. For a subset W of V, the *cut* induced by W, denoted by $\delta(W)$, is the set of edges $ij \in E$ with $i \in W$ and $j \notin W$. $\delta(W)$ is an *odd cut* if the number of odd nodes in W is odd, or equivalently, if $|\delta(W)|$ is odd.

Let $D = (V, A)$ be a digraph, and u and v be two nodes in V. A sequence $\omega = \langle (i_0, i_1), (i_1, i_2), \ldots, (i_{k-1}, i_k) \rangle$ of arcs in A with $i_0 = u$ and $i_k = v$ is a *diwalk* from u to v. ω is *open* if $u \neq v$ and *closed* otherwise; ω is a *dipath* if $i_s \neq i_t$ for $0 \leq s < t \leq k$. If ω is a dipath and $(i_k, i_0) \in A$, then the sequence $\langle (i_0, i_1), (i_1, i_2), \ldots, (i_{k-1}, i_k), (i_k, i_0) \rangle$ is a *dicycle*. D is *strongly connected* if for each pair of nodes $i, j \in V$, there are diwalks from i to j and from j to i.

Let $G = (V, E)$ be a graph and $ij \in E$. If we replace the edge ij by the arc (i, j), then we say that ij is *oriented* from i to j or that (i, j) is an *orientation of* ij. If $E_1 \subseteq E$ and A_1 is the set of arcs obtained by orienting each edge of E_1 in one of the two possible directions, then we call A_1 an *orientation of* E_1. If ω is a closed walk in $G = (V, E)$ and τ is a closed diwalk that is an orientation of ω, then τ is called a *diorientation* of ω.

Let $G = (V, E)$ be a graph and $c := (c_e)_{e \in E} \in \mathbb{R}^E$ be a *cost function* that assigns to each edge $e \in E$ a *cost* c_e. Then, for any $F \subseteq E$, we call $\sum_{f \in F} c_f$ the *cost of* F and denote it by $c(F)$. Similarly, for a digraph $D = (V, A)$, a cost function $c := (c_a)_{a \in A} \in \mathbb{R}^A$ and an arc set $B \subseteq A$, we denote the cost $\sum_{b \in B} c_b$ of B by $c(B)$. Let $G = (V, E)$ be a graph, $A := \{(i, j), (j, i) \mid ij \in E\}$ and $c := (c_{ij}, c_{ji})_{ij \in E} \in \mathbb{R}^A$, where c_{ij} resp. c_{ji} is the cost of edge ij when it is oriented from i to j resp. from j to i. (In the sequel, c_{ij} and c_{ji} will have these meanings whenever two numbers c_{ij} and c_{ji} are associated with an edge $ij \in E$.) In such a case, the cost of a subset E_1 of E is undefined, but the cost $c(A_1)$ of an orientation A_1 of E_1 is meaningful and is defined by $c(A_1) := \sum_{(i,j) \in A_1} c_{ij}$.

Using the above terminology, we can now define various types of postman problems.

The undirected postman problem (*The standard CPP*): Let $G=(V, E)$ be a connected graph with cost $c_e \in \mathbb{R}$ for each edge $e \in E$. A closed walk in G containing each edge of E at least once is called an *undirected postman tour* of G. The question is to find an undirected postman tour of G of minimum cost. (It is obvious that an undirected postman tour of finite minimum cost exists if and only if $c_e \geqslant 0$ for each edge $e \in E$.)

The directed postman problem: Let $D=(V, A)$ be a strongly connected digraph with cost $c_a \in \mathbb{R}$ for each arc $a \in A$. A closed diwalk in D containing each arc of A at least once is called a *directed postman tour* of D. The question is to find a directed postman tour of D of minimum cost. (It is obvious that a directed postman tour of finite minimum cost exists if and only if D contains no dicycle of negative cost.)

The mixed postman problem: Let $M=(V, E, A)$ be a mixed graph with cost c_e for each edge $e \in E$ and c_a for each arc $a \in A$. Suppose $D_M=(V, A')$ is the digraph with arc set $A' := A \cup \{(i, j), (j, i) \mid ij \in E\}$ and cost function $c' := (c'_a)_{a \in A'}$ defined by $c'_a := c_a$ for each arc $a \in A$ and $c'_{(i,j)} := c'_{(j,i)} := c_e$ for each edge $e := ij \in E$. A *mixed postman tour* of M is a closed diwalk in D_M containing each arc $a \in A$ at least once and containing one of the arcs (i, j) and (j, i) at least once for each edge $ij \in E$. The question is to find a mixed postman tour of M of minimum cost. (To guarantee existence of a feasible mixed postman tour, we assume that D_M is strongly connected, and to guarantee existence of a finite optimal mixed postman tour we assume that D_M contains no dicycle of negative cost.)

The windy postman problem: Let $G=(V, E)$ be a connected graph with a cost function $c := (c_{ij}, c_{ji})_{ij \in E}$. Here c_{ij} resp. c_{ji} is the cost of edge ij when it is oriented as (i, j) resp. (j, i). A diorientation of a closed walk in G containing each edge of E at least once is called a *windy postman tour* (short: *WP-tour*) of G. The question is to find a WP-tour of G of minimum cost. (It is obvious that a WP-tour of G of finite minimum cost exists if and only if the digraph $D=(V, A)$ with arc set $A := \{(i, j), (j, i) \mid ij \in E\}$ contains no dicycle of negative cost.)

Clearly, the undirected postman problem is a special case of the WPP. Moreover, since an arc $a=(i, j)$ with cost c_a can be transformed to an edge ij with costs $c_{ij}=c_a$ and $c_{ji}=\infty$, the directed and mixed postman problems can also be viewed as special cases of the WPP.

With respect to the computational complexity of these problems, it is known that the undirected and directed postman problems are solvable in polynomial time, cf. Edmonds (1965), Liebling (1970) and Edmonds and Johnson (1973). Some special cases of the mixed and windy postman problems can be solved in polynomial time, but they are NP-hard in general — see Edmonds and Johnson (1973), Guan (1984), Win (1989), and Papadimitriou (1976). In the literature very little can be found on the design and analysis of exact solution strategies for these NP-hard problems. As

far as we know, Christofides, Benavent, Campos, Corberan and Mota (1984) is the only computational study done for the mixed postman problem, and for the WPP no solution method with a computation analysis has been reported.

Before closing this section, let us mention some definitions of polyhedral theory. Suppose $a \in \mathbb{R}^n \setminus \{0\}$ and $\alpha \in \mathbb{R}$. Then $\{x \in \mathbb{R}^n \mid a^T x = \alpha\}$ is a *hyperplane* in $\mathbb{R}^n$ and $\{x \in \mathbb{R}^n \mid a^T x \leq \alpha\}$ is a half space in $\mathbb{R}^n$. A *polyhedron* in $\mathbb{R}^n$ is the intersection of finitely many half spaces in $\mathbb{R}^n$. A linear inequality $a^T x \leq \alpha$ with $a \in \mathbb{R}^n$, $\alpha \in \mathbb{R}$ is *valid* with respect to a polyhedron P, if $P \subseteq \{x \in \mathbb{R}^n \mid a^T x \leq \alpha\}$. A subset F of $\mathbb{R}^n$ is a *face* of a polyhedron P, if there exists a valid inequality $a^T x \leq \alpha$ for P such that $F = P \cap \{x \in \mathbb{R}^n \mid a^T x = \alpha\}$. In this situation we say that F is *induced* (*defined*) by the inequality $a^T x \leq \alpha$. A face of a polyhedron P is a *facet* if it is nonempty, different from P and maximal with respect to set inclusion.

3. Results on the windy postman polyhedron

In this section we summarize some polyhedral results on the WPP on which our cutting plane algorithm is based.

Let $G = (V, E)$ be the underlying graph of a WPP and τ a WP-tour of G. We define the *incidence vector* χ^τ of τ by $\chi^\tau := (\chi^\tau_{ij}, \chi^\tau_{ji})_{ij \in E}$, where χ^τ_{ij} resp. χ^τ_{ji} is the number of times (i, j) resp. (j, i) occurs in τ. The *windy postman polyhedron* (short: *WP-polyhedron*) of G, denoted by WP(G), is the convex hull of the incidence vectors of the WP-tours of G. Clearly, the WPP is equivalent to the linear program

$$\begin{aligned} &\min \quad c^T x \\ &\text{s.t.} \quad x \in \mathrm{WP}(G). \end{aligned} \tag{3.1}$$

Here $c := (c_{ij}, c_{ji})_{ij \in E}$ is the cost function of the given WPP.

To apply LP methods to (3.1), it is necessary to know a linear system describing WP(G) completely. But the NP-hardness of the WPP implies that it is unlikely that one can find such a linear system. However, recent computational studies in combinatorial optimization have shown that, quite frequently, even partial linear descriptions (of combinatorial polyhedra like WP(G)) are useful for the design of efficient LP-based algorithms for NP-hard problems for instance, see Padberg and Grötschel (1985), Grötschel and Holland (1991), and Padberg and Rinaldi (1991) for the travelling salesman problem, Grötschel, Jünger and Reinelt (1984) for the linear ordering problem, or Grötschel and Wakabayashi (1989) for a clustering problem etc.

With this motivation we have determined several classes of valid and facet defining inequalities of WP(G), cf. Win (1987), Grötschel and Win (1992). Some of these inequalities — those that are used in our algorithm — are listed in the following theorem:

Theorem 3.1. *Let $G=(V, E)$ be the underlying graph of a WPP.*

(a) *The following are valid equations and inequalities of* WP(G):

$$\sum_{ij\in\delta(i)} (x_{ij}-x_{ji})=0 \quad \textit{for all } i\in V, \tag{3.2}$$

$$x_{ij}+x_{ji}\geq 1 \quad \textit{for all } ij\in E, \tag{3.3}$$

$$x_{ij}\geq 0,\ x_{ji}\geq 0 \quad \textit{for all } ij\in E. \tag{3.4}$$

(b) *The equation system* (3.2) *describes the affine hull of* WP(G).

(c) *The dimension of* WP(G) *equals* $2|E|-|V|+1$.

(d) *An inequality of type* (3.3) *defines a facet of* WP(G) *if and only if ij is not a bridge of G.*

(e) *An inequality of type* (3.4) *defines a facet of* WP(G) *if and only if ij is not a bridge of G.*

(f) *For any odd cut $\delta(W)$ in G, the inequalities*

$$\sum_{ij\in\delta(W)} (x_{ij}+x_{ji})\geq |\delta(W)|+1, \tag{3.5}$$

$$\sum_{i\in W, j\in V\setminus W} x_{ij}\geq \frac{|\delta(W)|+1}{2}, \tag{3.6}$$

$$\sum_{i\in W, j\in V\setminus W} x_{ji}\geq \frac{|\delta(W)|+1}{2}, \tag{3.7}$$

are valid with respect to WP(G) *and they induce the same face. (In the sequel we will refer to these inequalities as odd cut inequalities.)*

(g) *An inequality of type* (3.5) *(and thus of type* (3.6) *or* (3.7)*) defines a facet of* WP(G) *if and only if the subgraphs of G induced by W and $V\setminus W$ are connected.* □

For the proof of this theorem, see Win (1987) or Grötschel and Win (1992). Let us define the polyhedra $P_1(G)$ and $P_2(G)$ by

$$P_1(G)=\{x:=(x_{ij}, x_{ji})_{ij\in E} \mid x \text{ satisfies (3.2)-(3.4)}\}$$

and

$$P_2(G)=P_1(G)\cap\{x \mid x \text{ satisfies (3.5) for each odd cut } \delta(W)\}.$$

Thus $\text{WP}(G)\subseteq P_2(G)\subseteq P_1(G)$. Moreover, it is easy to see that $\text{WP}(G)=\text{conv}\{x\in P_1(G) \mid x \text{ integral}\}$. Thus the following program is an integer programming formulation of the windy postman problem

$$\begin{aligned} &\min && c^{\mathrm{T}}x \\ &\text{s.t.} && x \text{ satisfies (3.2), (3.3), (3.4),} \\ &&& x \text{ integral.} \end{aligned} \tag{3.8}$$

It is shown in Win (1989) that $\text{WP}(G)=P_1(G)$ if and only if G is Eulerian. Therefore, for general graphs G, it seems reasonable to relax WP(G) to $P_2(G)$ (which contains

a large class of facets of type (3.5)) and to hope that the linear program

$$\begin{aligned} &\min \quad c^{\mathrm{T}}x \\ &\text{s.t.} \quad x \in P_2(G) \end{aligned} \tag{3.9}$$

provides a good lower bound on the length of an optimal WP-tour.

To check our belief and solve some real world problems, we have designed an LP-based cutting plane algorithm that optimizes the linear program (3.9). Our computational experiments of Section 5 show that $P_2(G)$ is a good relaxation of WP(G) indeed. Moreover, as our next theorem shows, $P_2(G)$ possesses another nice property.

Theorem 3.2. *The linear program* (3.9) *can be solved in time polynomial in the input length of the given WPP.*

Proof. Note that the number of inequalities of type (3.5) is in general exponential in $|V|+|E|$. However, it follows from the ellipsoid method — see Grötschel, Lovász and Schrijver (1981) — that (3.9) can be solved in polynomial time if and only if the *separation problem* for $P_2(G)$ (stated below) can be solved in polynomial time.

"Given a (rational) vector $x^* \in \mathbb{Q}^A$ (where $A = \{(i,j),(j,i) \mid ij \in E\}$), determine whether $x^* \in P_2(G)$ and if this is not the case find a hyperplane separating x^* from $P_2(G)$, i.e., find a vector $a(\neq 0) \in \mathbb{Q}^A$ and a number $\alpha \in \mathbb{Q}$ such that $a^{\mathrm{T}}x \geqq \alpha$ for every $x \in P_2(G)$ and $a^{\mathrm{T}}x^* < \alpha$."

The separation problem for $P_2(G)$ can be solved as follows. Given $x^* \in \mathbb{Q}^A$, one can check in polynomial time whether x^* satisfies the constraints (3.2)-(3.4) by direct substitution. We may thus assume that x^* satisfies all of them. For each edge $ij \in E$, define the weight $w_{ij} := x^*_{ij} + x^*_{ji} - 1$. Since, for any odd cut $\delta(W)$,

$$\begin{aligned} x^* \text{ satisfies (3.5)} &\Leftrightarrow \sum_{ij \in \delta(W)} (x^*_{ij} + x^*_{ji} - 1) \geqq 1 \\ &\Leftrightarrow \sum_{ij \in \delta(W)} w_{ij} \geqq 1, \end{aligned}$$

one can see that the separation problem for the odd cut inequalities (3.5) reduces to the problem of determining an odd cut of G of minimum weight with respect to the nonnegative weight function w. Padberg and Rao (1982) have shown that the latter is polynomially solvable. Hence the theorem is proved. □

4. Description of the algorithm

We now describe our algorithm to solve problem (3.9) resp. the windy postman problem. By Theorem 3.2, using the ellipsoid method, (3.9) can be solved in polynomial time. However, this procedure seems to be inefficient in practice because

of the rather poor (though polynomial) performance of the ellipsoid method. Therefore we have replaced the ellipsoid method by the simplex procedure in our algorithm. Thus our algorithm is theoretically nonpolynomial. Moreover, as mentioned in Section 1, it is not guaranteed to terminate with an optimal WP-tour. But the computational study of the next section shows that our algorithm works quite efficient in practice, i.e., in most of our test problems an optimal WP-tour was produced in reasonable computing time.

The algorithm is a (by now) standard cutting plane method, where an initial linear program is set up that is solved to optimality. One checks whether the optimum solution x^* represents a WP-tour and if not tries to find valid inequalities that are violated by x^*. These cutting planes are added to the current LP and the process is repeated. We will denote the polyhedron that is determined by the inequalities of the current LP by P and describe our code in more detail.

Step 1 (*Initialization*). Set $P := P_1(G)$.

This step initializes the polyhedron P with $P_1(G)$. As mentioned in (3.8), $P_1(G)$ has the desirable property that every integral point in $P_1(G)$ represents a WP-tour. Moreover, the number of constraints of the linear system describing $P_1(G)$ (not counting the nonnegativity constraints) is $|V|+|E|$. Linear programs of this type can be handled efficiently.

Step 2 (*Solving the LP*). Solve the linear program

$$\begin{aligned} &\min \quad c^T x \\ &\text{s.t.} \quad x \in P. \end{aligned} \tag{4.1}$$

For the solution of (4.1), we applied the simplex method. We used the linear programming package XMP of Roy Marsten, see Marsten (1981). The initial LP was solved by the XDUAL (dual simplex method) routine. The linear programs of later stages were always solved by the XDPH2 (phase-2 of the dual simplex method) routine, since a dual basic feasible solution is already known.

Step 3 (*Checking optimality*). Check whether the optimal LP solution, say x^*, of Step 2 represents a WP-tour. For this purpose we only have to check the integrality of x^* because the current polyhedron P is contained in $P_1(G)$ by construction. Thus, if x^* is integral then it represents an optimal WP-tour. We construct that tour and stop.

Step 4 (*Deleting nonbinding constraints*). From the linear system describing P, delete some odd cut inequalities which are nonbinding at the current LP solution x^*.

This step is parameter driven. Our code asks that a parameter ε is set to a small nonnegative number. Every odd cut inequality such that the value of the corresponding surplus variable is greater than ε is deleted. We usually chose ε between 0.1 and 0.3.

Step 5 (*Cutting plane recognition*). Find odd cut inequalities which are violated by x^*.

Such inequalities, if they exist, induce cutting planes which chop off x^*. How we did this job in our code is discussed in Sections 4.1 and 4.2.

Step 6 (*Adding cutting planes*). If some odd cut inequalities violated by x^* are found, add them to the linear system obtained in Step 4. Let P be the polyhedron determined by the resultant linear system and go to Step 2.

We observed in our experiments that the number of violated odd cut inequalities found at each iteration is relatively small. Thus, in our program, we added all inequalities we found.

Step 7 (*Constructing an approximate tour*). (We enter this step when the current LP solution is fractional and violates no odd cut inequality.) Apply a heuristic that uses the fractional LP solution x^* to construct an approximate WP-tour, calculate an upper bound of its relative error and stop.

The heuristic used in this step is described in Section 4.3.

4.1. *Cutting plane recognition: An exact algorithm*

Theorem (3.2) shows that the cutting plane recognition problem of Step 5 can be reduced to the minimum weighted odd cut problem which can be solved in polynomial time by the Padberg-Rao procedure. The worst-case performance of this exact algorithm is $O(|V|^4)$ and hence rather expensive. Therefore, in our code we first tried to determine violated odd cut inequalities by some fast heuristics and called the Padberg-Rao procedure only if the heuristics failed. In our application of the Padberg-Rao procedure, we construct a complete Gomory-Hu (or flow equivalent) tree and all violated odd cut inequalities which can be read from it are added to the current LP. The Padberg-Rao procedure calls a maximum flow algorithm as a subroutine; for this purpose we have used a max-flow code that is based on the Goldberg and Tarjan (1986) procedure.

4.2. *Cutting plane recognition: Some heuristics*

The heuristics we have applied are adaptations of those used by Grötschel and Holland (1985) in their cutting plane code for the perfect matching problem.

Heuristic 1. In this heuristic we construct a graph $G' = (V, E')$ with edge set $E' := \{ij \in E \mid x^*_{ij} + x^*_{ji} - 1 > 0\}$. Then we determine, by depth first search, odd components of G', i.e., components with an odd number of odd (with respect to G) nodes. If W is such a component, then obviously the odd cut inequality induced by W is violated by x^*. If there exists no such component of G', then the heuristic fails to recognize violated odd cut inequalities. The complexity of this heuristic is $O(|E'|)$. In practice, G' is very sparse and hence the complexity is $O(|V|)$ empirically.

Heuristic 2. In this heuristic we construct a graph $G' = (V, E')$ whose edge set $E' = \{ij \in E \mid x^*_{ij} + x^*_{ji} - 1 > \varepsilon\}$. Here ε is a small positive number which in our code was set to 0.2. As in Heuristic 1, we determine odd components of G' by depth first search, and for each odd component, we check whether it induces a violated odd cut inequality of x^*. If this is the case for some odd components, then we have found some cutting planes. Otherwise the heuristic fails to find odd cut inequalities violated by x^*. The complexity of this heuristic is also $O(|E'|)$ theoretically and $O(|V|)$ empirically.

In our code we call Heuristic 1 first and if it fails we turn to Heuristic 2.

4.3. A heuristic which constructs an approximate WP-tour

Let x^* be the fractional LP solution which is passed to Step 7 of our algorithm.

1. Construct the subgraph $G' = (V', E')$ of G that is induced by the edge set $E' := \{ij \in E \mid x^*_{ij}$ or x^*_{ji} is fractional$\}$.

(We have found in most of our experiments that G' is a small sparse graph compared with G.)

2. For each $i \in V'$, define $d'_i := \sum_{ij \in E'} (x^*_{ji} - x^*_{ij})$ and construct the linear program

$$\begin{aligned} \min \quad & \sum_{ij \in E'} (c_{ij} x_{ij} + c_{ji} x_{ji}) \\ \text{s.t.} \quad & \sum_{ij \in E'} (x_{ji} - x_{ij}) = d'_i \quad \text{for all } i \in V', \\ & x_{ij} + x_{ji} \geqslant 1 \quad \text{for all } ij \in E', \\ & x_{ij} \geqslant 0,\ x_{ji} \geqslant 0 \quad \text{for all } ij \in E'. \end{aligned} \tag{4.2}$$

(One can see that the d'_i's are integers, $\sum_{i \in V'} d'_i = 0$, and that $(x^*_{ij}, x^*_{ji})_{ij \in E'}$ is a fractional feasible solution of (4.2).)

3. Solve the linear program (4.2) (by the simplex method) to obtain an optimal vertex solution, say $x' := (x'_{ij}, x'_{ji})_{ij \in E'}$.

4. (In this step we will use the following polyhedral result of Win (1987). "Let $y := (y_{ij}, y_{ji})_{ij \in E'}$ be a vertex of the polyhedron of the LP (4.2). Then (i) each component of y is either $\frac{1}{2}$ or integral and (ii) $y_{ij} = \frac{1}{2}$ implies $y_{ji} = \frac{1}{2}$.")

Define a vector $\bar{x} = (\bar{x}_{ij}, \bar{x}_{ji})_{ij \in E'}$ by $\bar{x}_{ij} = x'_{ij}$ if x'_{ij} is integral and $\bar{x}_{ij} = 1$ otherwise (i.e., if $x'_{ij} = \frac{1}{2}$).

(It is easy to see that $\bar{x}$ is an integral feasible solution of (4.2) and $\bar{x}$ together with $(x^*_{ij}, x^*_{ji})_{ij \in E \setminus E'}$ gives a feasible WP-tour of the given WPP.)

5. Combine $\bar{x}$ and $(x^*_{ij}, x^*_{ji})_{ij \in E \setminus E'}$ to get a new vector $\hat{x}$, construct the WP-tour represented by $\hat{x}$ and stop.

Roughly speaking, the idea of this heuristic is to fix the integral components of the last fractional solution x^* and then to reasonably transform its fractional components to integers so that they together with the fixed integral components constitute a good feasible WP-tour. We have found that our heuristic works very well in practice and generates good approximate solutions which are almost optimal. The cost of the last fractional solution x^* (which is usually a tight lower bound of the cost of an optimal WP-tour) is also useful for the evaluation of the quality of the generated approximate WP-tour.

5. Computational results

The practical performance of our algorithm has been studied with several test problems. As the underlying graphs of these problems, we chose six real world transportation networks. The number of nodes in these graphs ranges from 52 to 264; the number of edges from 78 to 489. Our computational study is composed of three parts. The first one deals with WPPs, the second one with mixed postman problems and the last one with standard CPPs. Clearly our major interest is in the first two parts that concern NP-hard problems.

We have coded our algorithm in FORTRAN and have run it on the NORSK DATA ND-540 of the University of Augsburg under the operating system SINTRAN III-VSX/500. The CPU times reported are for the execution of the entire run including the input/output operations, overhead etc. Fractions of seconds have been rounded up.

5.1. Experiments with WPPs

As mentioned above, six real world networks and the corresponding real world edge costs (lengths) were chosen for our computational study. The real world edge costs were symmetric, i.e., $c_{ij} = c_{ji}$ for each edge ij, and hence not suitable to use in our test WPPs. Thus we generated asymmetric edge costs randomly. We used two random cost generation procedures described in Sections 5.1.1 and 5.1.8.

5.1.1. A random cost generation procedure

Let $G = (V, E)$ be a real world graph with real world (symmetric) integer costs c'_{ij} for each edge ij in E. Suppose $|V| = n$, the nodes of V are indexed by the integers $1, 2, \ldots, n$, and for each node $i \in V$, $N(i)$ denotes the set of nodes adjacent to i. A

procedure which we used to generate random costs c_{ij} and c_{ji} for each edge $ij \in E$ is as follows.

Procedure 5.1.

For $i := 1$ to n do.

For each $j \in N(i)$ do.

Begin.

Use a random generator R to generate an integer k in an interval $[-a, a]$. (Here a is an arbitrary but fixed positive integer.)

Set $c_{ij} := c'_{ij} + k$.

If $c_{ij} \leqslant 0$ then set $c_{ij} := 1$.

End.

By choosing small intervals $[-a, a]$, this procedure models real world problems where the difference between c_{ij} and c_{ji} is not too large. Clearly, the random costs generated by Procedure 5.1 depend on the random generator R as well as on the order of the nodes in the adjacency lists $N(i)$. A complete documentation of our choices can be found in Win (1987). For the real world (symmetric) edge costs of our graphs, we refer to the original documentations of the problems. The graphs we have chosen can be found in Win (1987), or in the original documentations.

5.1.2. WPPs with underlying graph G_1

G_1 is the underlying graph of a standard CPP which, together with real world edge costs, was originally given in Burkard and Derigs (1980, p. 98). It contains 52 nodes and 78 edges. The average of the real world edge costs of this graph is 57.08. (We give this information so as to have an idea about the perturbation introduced by Procedure 5.1.) Taking G_1 as the underlying graph, we created five WPPs, i.e., we generated five sets of edge costs by Procedure 5.1. Each set of these edge costs is uniquely determined by the interval $[-a, a]$ and the initial value S set to the variable SEED of the random generator R. Therefore we can identify each of our test problems by the corresponding pair $(S, [-a, a])$. Table 1 shows the identifiers of these problems and the corresponding computational results. All problems were solved to optimality, and all cuts were produced by our cutting plane recognition heuristics (see Section 4.2). In that table and also in the others, the column labels have the following meanings:

PRO: problem identifier;

LP: number of LPs solved (=number of simplex calls);

CONS: minimum and maximum number of constraints (excluding the non-negativity constraints) of the LPs solved (we note here that the number of variables, not counting slack and surplus ones, of these LPs equals $2|E|$);

CUT: total number of cuts recognized;

HCUT: number of cuts recognized by heuristics;

ERR: an upper bound of the relative error, expressed in percentage, of the tour generated by the algorithm (this upper bound is defined by $100\,(\hat{C} - C')/C'$, where

$\hat{C}$ is the cost of the tour generated by the algorithm and C' is a lower bound of the cost of an optimal tour, upper bound 0% means that the generated tour is optimal);

COST: cost of the tour generated by the algorithm;

CPU: CPU time in min:sec on the NORSK DATA ND-540;

PRP: number of times the Padberg-Rao procedure is called, and the percentage of the total time spent in this procedure;

ARC: number of arcs in the tour produced by the algorithm.

Table 1

PRO	LP	CONS	CUT	HCUT	ERR	COST	CPU	PRP	ARC
(3, [−5, 5])	7	130-179	56	56	0	6635	0:13	0, 0	104
(4, [−5, 5])	6	130-179	54	54	0	6656	0:12	0, 0	104
(3, [−8, 8])	6	130-179	54	54	0	6602	0:12	0, 0	104
(4, [−8, 8])	7	130-180	56	56	0	6629	0:17	0, 0	105
(3, [−10, 10])	7	130-180	56	56	0	6582	0:13	0, 0	104

5.1.3. WPPs with underlying graph G_2

G_2 is the transportation network of a waste collection problem. It contains 101 nodes and 185 edges. For G_2 itself and coordinates of its nodes, see Appendix A14 and A18 of Paessens (1981) respectively. From the node coordinates, we calculate the Euclidean distances between adjacent nodes, round them to the nearest integers and take the results as the real world costs. We use Procedure 5.1 to generate five WPPs. The average of these real world edge costs is 29.43. The identifiers of these problems and corresponding computational results are given in Table 2. We can see that our algorithm produced optimal WP-tours for four problems and for the remaining one it generated an approximate WP-tour whose deviation from the optimal value is not more than 1.37%. The exact cutting plane recognition algorithm was called in two cases.

Table 2

PRO	LP	CONS	CUT	HCUT	ERR	COST	CPU	PRP	ARC
(1, [−5, 5])	20	286-383	112	112	0	5998	1:35	0, 0	219
(2, [−5, 5])	26	286-391	125	120	0	5900	1:40	1, 0.30	219
(1, [−8, 8])	28	286-387	128	128	0	5904	1:49	0, 0	219
(2, [−8, 8])	23	286-394	132	118	1.37	5817	1:33	2, 0.57	220
(1, [−10, 10])	23	286-385	123	123	0	5842	1:43	0, 0	219

5.1.4. WPPs with underlying graph G_3

G_3 is a real world transportation network containing 110 nodes and 193 edges. G_3 itself and coordinates of its nodes can be seen in Appendix A15 and A19 of Paessens (1981) respectively. The average of the real world edge costs of this graph is 29.52.

As in the case of G_2, we constructed five test problems with G_3 as the underlying graph. Table 3 collects the identifiers of these problems and corresponding computation results. We found an optimal WP-tour for each test problem. The exact cutting plane recognition algorithm was called in two cases.

Table 3

PRO	LP	CONS	CUT	HCUT	ERR	COST	CPU	PRP	ARC
(3, [−5, 5])	16	303–398	104	104	0	6611	1:15	0, 0	234
(4, [−5, 5])	14	303–407	120	101	0	6578	1:13	2, 0.59	233
(3, [−8, 8])	17	303–405	116	105	0	6524	1:22	1, 0.26	234
(4, [−8, 8])	13	303–398	102	102	0	6469	1:07	0, 0	233
(3, [−10, 10])	26	303–400	120	120	0	6466	1:35	0, 0	234

5.1.5. WPPs with underlying graph G_4

G_4 is the transportation network of a waste collection problem. It contains 172 nodes and 247 edges. For G_4 itself and the coordinates of its nodes, see Appendix A16 and A20 of Paessens (1981), respectively. The average of the real world edge costs of this graph is 24.11.

As before, we constructed five test problems with G_4 as the underlying graph. The identifiers of these problems and corresponding computation results are listed in Table 4. For each problem our algorithm produced an optimal WP-tour. The exact cutting plane recognition algorithm was called only in one case.

Table 4

PRO	LP	CONS	CUT	HCUT	ERR	COST	CPU	PRP	ARC
(5, [−5, 5])	18	419–545	150	150	0	6947	2:26	0, 0	313
(6, [−5, 5])	23	419–553	162	162	0	7026	2:31	0, 0	311
(5, [−8, 8])	15	419–549	150	150	0	6811	2:20	0, 0	312
(6, [−8, 8])	22	419–551	158	158	0	6917	2:26	0, 0	311
(5, [−10, 10])	16	419–543	150	145	0	6704	2:48	1, 0.35	312

5.1.6. WPPs with underlying graph G_5

G_5 is a real world graph which, together with real world edge costs, can be found in Alewell (1980, p. 271). In the original graph, an intersection of two or more edges (roads) is taken as a node (and given a node number) only when there is a customer at that position. For our experiment we assumed all such intersections also as nodes and assigned node numbers to them. Our modified graph, shown in Win (1987), contains 179 nodes and 307 edges. The average of the real world edge costs of this graph is 3.94. As before, we created five problems with G_5 as the underlying graph. The identifiers of these problems and corresponding computational statistics are given in Table 5. For four problems our algorithm produced optimal WP-tours and,

for the remaining one, it generated an approximate tour whose relative error is not more than 5.16%. The exact cutting plane recognition algorithm was called in two cases.

Table 5

PRO	LP	CONS	CUT	HCUT	ERR	COST	CPU	PRP	ARC
(5, [−5, 5])	14	486-620	172	172	0	1277	3:11	0, 0	391
(6, [−5, 5])	92	486-717	476	269	0	1265	16:55	13, 3.34	390
(5, [−8, 8])	9	486-609	144	144	0	1338	2:26	0, 0	396
(6, [−8, 8])	70	486-715	490	209	5.16	1355	11:56	15, 5.45	395
(5, [−10, 10])	7	486-603	138	138	0	1391	2:23	0, 0	398

5.1.7. WPPs with underlying graph G_6

Unlike the preceding ones, G_6 is not extracted from the literature. It is the underlying graph of a real world multiple travelling salesman problem which was studied at the University of Augsburg in 1987. It contains 264 nodes and 489 edges. The real world edge costs and adjacency lists are available from the authors. The average of the real world edge costs of this graph is 13.35. As before, we created five WPPs on G_6. Table 6 describes the identifiers of these problems and the associated computational results. For three problems we found optimal tours and for the other two near optimal solutions. The exact cutting plane recognition algorithm was called in each case.

In the 30 runs documented, an optimal WP-tour was found in 26 cases. The exact cutting plane recognition procedure was needed only in 12 cases.

Table 6

PRO	LP	CONS	CUT	HCUT	ERR	COST	CPU	PRP	ARC
(1, [−5, 5])	29	753-1025	338	261	3.24	7286	13:09	4, 1.11	601
(2, [−5, 5])	35	753-1008	319	271	0	6873	10:58	2, 0.48	591
(1, [−8, 8])	36	753-1035	343	278	2.79	7043	12:59	5, 0.85	611
(2, [−8, 8])	50	753-1015	354	281	0	6587	17:40	3, 0.58	595
(1, [−10, 10])	47	753-1025	370	285	0	6728	13:27	3, 0.83	598

5.1.8. An additional experiment with WPPs

In the above 30 test problems, edge costs are generated by Procedure 5.1 and hence they are influenced by the real world costs. We also studied the performance of our algorithm on problems with purely randomly generated costs. We fixed integers $a < b$, and used a random generator to produce, for each edge, integral edge costs in the interval $[a, b]$. The edge costs generated by this procedure are uniquely determined by the interval $[a, b]$ and the initial value S set to the variable SEED of the random generator R. Therefore a problem with underlying graph G_i and

edge costs generated from the interval $[a, b]$ by setting the initial value S to the variable SEED can be identified by the triplet $(G_i, S, [a, b])$.

Using the above procedure we created six problems with underlying graphs G_1–G_6 and integer edge costs generated from the intervals [1, 100], [1, 200] and [1, 300]. The identifiers of these problems and corresponding computation results are listed in Table 7. Optimal WP-tours were obtained for all but one problem.

Table 7

PRO	LP	CONS	CUT	HCUT	ERR	COST	CPU	PRP	ARC
$(G_1, 1, [1, 100])$	5	130–178	50	50	0	4629	0:13	0, 0	108
$(G_2, 2, [1, 100])$	8	286–356	78	78	0	8117	0:56	0, 0	219
$(G_3, 1, [1, 200])$	11	303–381	88	88	0	19491	0:59	0, 0	241
$(G_4, 2, [1, 200])$	11	419–548	148	142	0	25450	2:03	1, 0.45	314
$(G_5, 1, [1, 300])$	41	486–711	351	197	8.68	50452	8:40	9, 3.37	404
$(G_6, 2, [1, 300])$	44	753–1007	357	252	0	66444	14:26	3, 1.13	602

5.1.9. Some remarks on the computational results

The computational results with 36 test problems are described in Table 1–7. Of these problems, our algorithm produced optimal WP-tours in 31 cases. WP-tours very near to optimal ones were found in the other cases. This empirically shows that $P_2(G)$ is a "nice" relaxation of WP(G) indeed.

One can notice in the tables that most of the cutting planes were recognized by heuristics and that, for many problems, we reached the optimum without calling the Padberg–Rao exact separation procedure at all. Thus our cutting plane recognition heuristics work quite well in practice. The maximum number of cutting planes was always recognized in the first iteration (it is nearly the number of odd nodes of the underlying graph). In the later iterations we recognized relatively few cutting planes. The number of nonbinding constraints deleted in each iteration was also rather small.

One can see in the tables that computation times required by our algorithm are also reasonable. We found in all our test problems that about 90% of the overall running time was spent for solving the linear programs. Thus a significant improvement of the LP-code we use will imply the same for our algorithm. Nearness of the approximate tours (if they were produced) to optimal solutions also shows the efficiency of the heuristic of Section 4.3.

To provide the reader with a picture of the detailed computational statistics with our test problems, we have chosen the one with underlying graph G_5 and identifier $(5, [-8, 8])$. As Table 5 shows, we had to solve 9 LPs until we reached the optimum of that problem. In Table 8 we describe the number of cutting planes deleted in each iteration (abbreviated as NCD), the number of cutting planes added in each

Table 8

Iteration No.	NCD	NCA	TNC	COLP
1	0	0	486	1183.50
2	0	86	572	1302.25
3	4	18	586	1322.25
4	7	14	593	1330.50
5	1	12	604	1335.00
6	4	6	606	1338.00
7	1	2	607	1338.00
8	2	4	609	1338.00
9	5	2	606	1338.00

iteration (abbreviated as NCA), total number of constraints (excluding the nonnegativity ones) of the LP of each iteration (abbreviated as TNC) and the cost of the optimal LP solution of each iteration (abbreviated as COLP).

One can notice that in the third iteration we are already very close to the optimum value and that, starting at the sixth iteration, the optimal LP solution value is equal to the cost of an optimal WP-tour. But the integral solution is obtained only in the ninth iteration. The distribution of running time of this problem to each task of our algorithm is as follows:

LP solving: 95.72%,
LP updating: 0.90%,
cut recognition heuristics: 1.28%
Padberg-Rao procedure: 0.00%,
reading input data and initial LP set-up: 1.40%,
WP-tour construction: 0.70%.

5.2. *Experiments with mixed postman problems*

We remarked in Section 2 that one can transform a mixed postman problem into a WPP by replacing an arc $a := (i, j)$ of cost c_a by an edge ij of costs $c_{ij} := c_a$, $c_{ji} := \infty$. Thus one can apply our algorithm also to mixed postman problems. To see how our algorithm works on these problems, we constructed 8 mixed graphs by assigning some orientations to some edges of the graphs G_1-G_4. As edge and arc costs of these mixed graphs, we have used the real world costs. Table 9 shows the computational results with these problems. In that table, the problem identifier (G_i, a, b, c) states that the mixed graph is obtained from the (undirected) graph G_i and that it contains a nodes, b edges and c arcs. For the detailed structure of these mixed graphs, see Win (1987). One can see in Table 9 that, for each problem, our algorithm produced an optimal solution.

5.3. *Experiments with standard CPPs*

Since standard CPPs can also be viewed as WPPs, we can apply our algorithm also to them. To see how our algorithm works in this case, we have tested it with six

Table 9

PRO	LP	CONS	CUT	HCUT	ERR	COST	CPU	PRP	ARC
(G_1, 52, 47, 31)	3	130–161	32	32	0	7065	0:09	0, 0	113
(G_1, 52, 37, 41)	7	130–168	42	42	0	6879	0:12	0, 0	114
(G_2, 101, 122, 63)	13	286–350	74	74	0	6759	0:51	0, 0	242
(G_2, 101, 95, 90)	20	286–350	84	84	0	8382	0:57	0, 0	282
(G_3, 110, 120, 73)	19	303–391	112	94	0	7306	1:09	3, 1.17	251
(G_3, 110, 101, 92)	9	303–357	58	58	0	8615	0:42	0, 0	283
(G_4, 172, 154, 93)	11	419–498	96	96	0	9451	1:19	0, 0	396
(G_4, 172, 131, 116)	10	419–514	118	118	0	7645	1:24	0, 0	329

Table 10

PRO	LP	CONS	CUT	HCUT	ERR	COST	CPU	PRP	ARC
(G_1, 52, 78)	7	130–181	56	56	0	6700	0:14	0, 0	104
(G_2, 101, 185)	19	286–385	114	110	0	6171	1:25	1, 0.32	219
(G_3, 110, 193)	18	303–402	110	110	0	6784	1:22	0, 0	233
(G_4, 172, 247)	28	419–563	173	173	0	7217	3:14	0, 0	312
(G_5, 179, 307)	21	486–651	192	192	0	1413	3:45	0, 0	374
(G_6, 264, 489)	17	753–973	254	254	0	7494	7:42	0, 0	585

real world CPPs. These problems are obtained from the aforementioned graphs G_1–G_6 and associated real world edge costs. We know that our algorithm is neither polynomial nor exact. But, to our surprise, our algorithm produced optimal tours for all problems. The computation times required are also quite modest, see Table 10. In that table, the problem identifier (G_i, a, b) states that the underlying graph of the CPP is G_i which contains a nodes and b edges.

6. Final remarks

In this paper we have described a cutting plane algorithm and its implementation for the windy postman problem which is based on a partial linear description of the WP-polyhedron. This algorithm can also be applied to mixed, undirected, and directed postman problems. The practical performance of our algorithm has been investigated with several test problems defined on graphs of up to 264 nodes and 489 edges. Our computational experiments of the previous section show that out of 36 test-WPPs our algorithm produced optimal WP-tours in 31 cases and good approximate tours in the others. In the cases of mixed postman and standard Chinese postman problems, the algorithm produced optimal solutions of all test problems. Moreover, we found that the computation times required are quite reasonable. Therefore we believe that our algorithm is efficient enough to be used in practice.

To our knowledge, there is no computational study of an exact method for the windy postman problem in the literature. In the case of the mixed postman problem Christofides, Benavent, Campos, Corberan and Mota (1984) is the only study we know. It proposes a branch and bound based combinatorial algorithm for the mixed postman problem and reports computational results with graphs of up to 50 nodes, 85 arcs and 36 edges. However, we were not able to test our code on these problems.

The computational experiments reported above show that our algorithm works quite well in practice. But further significant improvements may still be achieved. We mention a few possibilities. Firstly, one can add an enumeration phase like branch and bound to our algorithm so that optimality can always be guaranteed. Secondly, further classes of cutting planes should be considered. We have based our code on odd cut inequalities only although we know additional classes of valid and facet defining inequalities for the WP-polyhedron, see Win (1987) and Grötschel and Win (1992). However, we do not have polynomial time separation procedures for these classes. Fast separation heuristics, though, will probably help to improve the code, in particular, to reduce the number of calls of the LP-solver.

References

K. Alewell, *Standort und Distribution: Entscheidungsfälle* (Gabler, Wiesbaden, 1980).

A. Bachem and M. Grötschel, "New aspects of polyhedral theory," in: B. Korte, ed., *Modern Applied Mathematics: Optimization and Operations Research* (North-Holland, Amsterdam, 1982) pp. 51–106.

J.A. Bondy and U.S.R. Murty, *Graph Theory with Applications* (American Elsevier, New York, and Macmillan, London, 1976).

R. Burkard and U. Derigs, *Assignment and Matching Problems: Solutions Methods with FORTRAN Programs*, Lecture Notes in Economics and Mathematical Systems No. 184 (Springer, Berlin, 1980).

N. Christofides, E. Benavent, V. Campos, A. Corberan and E. Mota, "An optimal method for the mixed postman problem," in: P. Thoft-Christensen, ed., *System Modelling and Optimization*, Lecture Notes in Control and Information Sciences No. 59 (Springer, Berlin, 1984).

J. Edmonds, "The Chinese postman problem," *Operations Research* 13, Supplement 1 (1965) B-73.

J. Edmonds and E.L. Johnson, "Matching, Euler tours and the Chinese postman," *Mathematical Programming* 5 (1973) 88–124.

A.V. Goldberg and R.E. Tarjan, "A new approach to the maximum flow problem," *Proceedings of the 18th ACM Symposium on Theory of Computing* (1986) pp. 136–146.

M. Grötschel and O. Holland, "Solving matching problems with linear programming," *Mathematical Programming* 33 (1985) 243–259.

M. Grötschel and O. Holland, "Solution of large-scale symmetric travelling salesman problems," *Mathematical Programming* 51 (1991) 141–202.

M. Grötschel, M. Jünger and G. Reinelt, "A cutting plane algorithm for the linear ordering problem," *Operations Research* 32 (1984) 1195–1220.

M. Grötschel, L. Lovász and A. Schrijver, "The ellipsoid method and its consequences in combinatorial optimization," *Combinatorica* 1 (1981) 169–197.

M. Grötschel and Y. Wakabayashi, "A cutting plane algorithm for a clustering problem," *Mathematical Programming (Series B)* 45 (1989) 59–96.

M. Grötschel and Z. Win (1992), "On the windy postman polyhedron," to appear.

M. Guan, "Graphic programming using odd or even points," *Chinese Mathematics* 1 (1962) 273–277.

M. Guan, "On the windy postman problem," *Discrete Applied Mathematics* 9 (1984) 41–46.

T.M. Liebling, *Graphentheorie in Planungs- und Tourenproblemen*, Lecture Notes in Operations Research and Mathematical Systems No. 21, (Springer, Berlin, 1970).

R. Marsten, "The design of the XMP linear programming library," *ACM Transactions on Mathematical Software* 7 (1981) 481-497.

M.W. Padberg and M. Grötschel, "Polyhedral computations," in: E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D. Shmoys eds., *The Traveling Salesman Problem* (Wiley, Chichester, 1985) pp. 307-360.

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

M. Padberg and G. Rinaldi, "A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems," *SIAM Review* 33 (1991) 60-100.

H. Paessens, "*Tourenplanung bei der regionalen Hausmüllentsorgung*," Institut für Siedlungswasserwirtschaft, Universität Karlsruhe (Karlsruhe, 1981).

C.H. Papadimitriou, "On the complexity of edge traversing," *Journal of the Association for Computing Machinery* 23 (1976) 544-554.

Z. Win, "Contributions to routing problems," Ph. D. Thesis, Universität Augsburg (Augsburg, 1987).

Z. Win, "On the windy postman problem on Eulerian graphs," *Mathematical Programming* 44 (1989) 97-112.