Available online at www.sciencedirect.com
ScienceDirect
European Journal of Operational Research 174 (2006) 1396–1413
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/ejor

Continuous Optimization

A smoothing heuristic for a bilevel pricing problem

Jean-Pierre Dussault [a], Patrice Marcotte [b], Sébastien Roch [c], Gilles Savard [d,*]

[a] *Département de Mathématiques et d'Informatique, Université de Sherbrooke, Sherbrooke, Qué., Canada*
[b] *Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, Montréal, Qué., Canada*
[c] *Department of Statistics, University of California, Berkeley, CA, USA*
[d] *Département de Mathématiques et de Génie Industriel, École Polytechnique de Montréal, C.P. 6079, Succ. Centre-ville, Montréal, Qué., Canada*

Received 5 January 2004; accepted 1 July 2004
Available online 1 September 2005

Abstract

In this paper, we provide a heuristic procedure, that performs well from a global optimality point of view, for an important and difficult class of bilevel programs. The algorithm relies on an interior point approach that can be interpreted as a combination of smoothing and implicit programming techniques. Although the algorithm cannot guarantee global optimality, very good solutions can be obtained through the use of a suitable set of parameters. The algorithm has been tested on large-scale instances of a network pricing problem, an application that fits our modeling framework. Preliminary results show that on hard instances, our approach constitutes an alternative to solvers based on mixed 0–1 programming formulations.

Keywords: Global optimization; Bilevel programming; Interior point methods; Smoothing; Implicit programming; MPEC; Complementarity; Network pricing

1. Introduction

Our aim in this work is to design an efficient algorithm for solving a class of bilevel programs upon which pricing and revenue management problems have been based (see [9,22,23]). Bilevel programming, or mathematical programming with equilibrium constraints (MPEC), is a branch of mathematics concerned with the optimization of an objective function over the solution set of some mathematical program [32,33]. It

* Corresponding author. Tel.: +1 5143404765; fax: +1 5143404463.
E-mail addresses: jean-pierre.dussault@dmi.usherb.ca (J.-P. Dussault), marcotte@iro.umontreal.ca (P. Marcotte), sroch@stat.berkeley.edu (S. Roch), gilles.savard@polymtl.ca (G. Savard).

0377-2217/$ - see front matter
doi:10.1016/j.ejor.2004.07.076

is closely related to Stackelberg games, where a leader incorporates within her decision process the reaction of the followers to her course of action. It can be fairly argued that all decision-making processes involving variables that are not in the direct control of the leader fit this framework. Examples of such instances abound in industry or government, where the impact of policies on customers should be evaluated before they are implemented. Its mathematical formulation is:

$$(BP) \quad \min_{(u,v)} F(u,v)$$
$$\begin{cases} u \in U \subseteq \mathbb{R}^{n_u}, \\ v \in \arg\min_{v \in V(u) \subseteq \mathbb{R}^{n_v}} f(u,v), \end{cases}$$

where u and v are respectively, the upper and lower-level variables. It is well known that bilevel programming is intractable (see e.g. [19,34,1]), yielding different research avenues. One trend focuses on exact algorithms. Indeed, it is sometimes possible to transform a bilevel program into a (one-level) mixed 0–1 program (see e.g. [1]). Although this has the advantage that general-purpose solvers can be used, this framework tends to break down on large instances.

Another line of attack consists in replacing the follower's optimization problem, whenever it is convex, by its stationarity conditions. This yields a one-level program with complementarity constraints, known as MPEC [25]. The complementarity constraints, that hide the combinatorial structure of the problem, are particularly difficult to handle. Moreover, MPECs are ill-structured in the sense that they generally satisfy no constraint qualification, and that stationarity can fail to be characterized by a Karush–Kuhn–Tucker system, and that general-purpose NLP solvers may fail to uncover such local minima or even mere stationary points.[1] Different approaches have been developed to solve MPECs, and we mention three that are related to our line of attack. First, the implicit programming approach (see e.g. [26,25,11]) can be applied when the follower's choice is unique for every decision made by the leader. It usually leads to the use of nonsmooth analysis techniques [8]. Next, in the smoothing approach [12,16,17,20,30], one reformulates the complementarity constraints as nondifferentiable constraints and then applies smoothing techniques on these constraints.[2] A third approach is provided by the algorithm PIPA of Luo et al. [25], which is based on interior-point methods (see however [24]). From a practical point of view, one drawback of these techniques is that they guarantee local optimality, at best, under a variety of strong assumptions.

Bilevel programming is particularly suited at modeling pricing problems. In this paper, we focus on a network pricing problem (MaxToll in the sequel) that has been defined and analyzed by Labbé et al. [22,23]. In this model, the follower consists in travelers moving between their respective origin and destination, using only shortest paths. At the upper level, the leader has the power to levy tolls on a subset of road segments, and aims at maximizing his revenue. The leader's dilemma is to avoid tolls too low—because they produce low revenue—, and tolls too high—because they urge the network users to choose toll-free paths. It was recently shown that MaxToll, even in its simplest form, is strongly NP-hard [27].[3]

Besides toll optimization, variants of the basic model can be useful for pricing purposes in the airline and telecommunication industries [9]. See also [2] for a related model with applications to tax credits in biofuel production. Actually, our algorithm can be applied to a much larger class of problems, i.e., bilevel programs with bilinear objectives and linear constraints.

In this paper, our aim is to combine the implicit programming and smoothing techniques, and design an algorithm that is both efficient on large realistic instances and performs well from a global optimality point of view. Our choice to study the special case of MaxToll has been motivated both by its practical

[1] Note however that recent attempts at tackling MPECs with general NLP solvers such as SQP solvers [14,15] or interior-point solvers [3] have been successful, to some extent.

[2] This technique originated in works on complementarity problems. For references, see [5,13].

[3] The classic reference on NP-hardness is [18].

importance and by the availability of exact solvers that are useful for numerical comparisons. Note that, for MaxToll, the issue of local convergence is to some extent irrelevant, in the sense that locally optimal toll schedules can easily be derived from the knowledge of lower level basic solutions.

The rest of the paper is organized as follows: following preliminaries in Section 2, the algorithm is presented in Section 3 and numerical results are detailed and analyzed in Section 4.

2. Problem formulation and preliminaries

2.1. Problem formulation

Following [22,23], we specialize (BP) to a formulation suited for pricing or taxation problems, where the objective functions are bilinear and constraints are linear. To be more specific, we assume that a network structure with a set $\mathscr{K}$ of commodities underlies the problem (but this is not necessary). One may think of $|\mathscr{K}|$ classes of users[4] going from s^k to t^k, $k \in \mathscr{K}$, in the network. The users travel on shortest paths and the leader maximizes its profit by appropriately setting tolls on a subset of arcs. The resulting bilevel program, which we refer to as MaxToll, is

$$\text{MaxToll} \quad \max_{t,x^k,y^k} t' \sum_{k\in\mathscr{K}} x^k$$
$$\begin{cases} \min_{x^k,y^k} (c+t)' \sum_{k\in\mathscr{K}} x^k + d' \sum_{k\in\mathscr{K}} y^k \\ \forall k \in \mathscr{K} \begin{cases} A_1 x^k + A_2 y^k = b^k, \\ x^k, y^k \geqslant 0, \end{cases} \end{cases}$$

where $t \in \mathbb{R}^{m_x}$ is the toll vector (the control variable), $x^k \in \mathbb{R}^{m_x}$ is the flow of commodity k on toll arcs and $y^k \in \mathbb{R}^{m_y}$ is the flow of commodity k on toll-free arcs. The vectors $c \in \mathbb{R}^{m_x}$ and $d \in \mathbb{R}^{m_y}$ are fixed costs on tolled and toll-free arcs, respectively, and the linear constraints at the lower level are flow conservation constraints characterized by the node-arc incidence matrix (A_1, A_2) and the commodity demand vectors b^k, i.e.,

$$b_i^k = \begin{cases} n_k & \text{if } i \text{ is the origin node of commodity } k, \\ -n_k & \text{if } i \text{ is the destination node of commodity } k, \\ 0 & \text{otherwise.} \end{cases}$$

(See Fig. 1 for an example).

Of course, one could also have introduced coupling constraints (e.g. capacity constraints), fixed costs, commodity dependent tolls (e.g. users having a different perception of time and money) for modeling purposes.

To simplify notation, we define $A = [A_1 \quad A_2]$, $C(t) = (c' + t', d')'$, $z^k = (x^{k'}, y^{k'})'$, $x = \sum_{k\in\mathscr{K}} x^k$ and $z = \sum_{k\in\mathscr{K}} z^k$. This yields the equivalent program

$$\max_{t,z^k} t'x$$
$$\begin{cases} \min_{z^k} C(t)'z \\ \forall k \in \mathscr{K} \begin{cases} Az^k = b^k, \\ z^k \geqslant 0. \end{cases} \end{cases}$$

[4] We denote the cardinality of a set S by $|S|$.

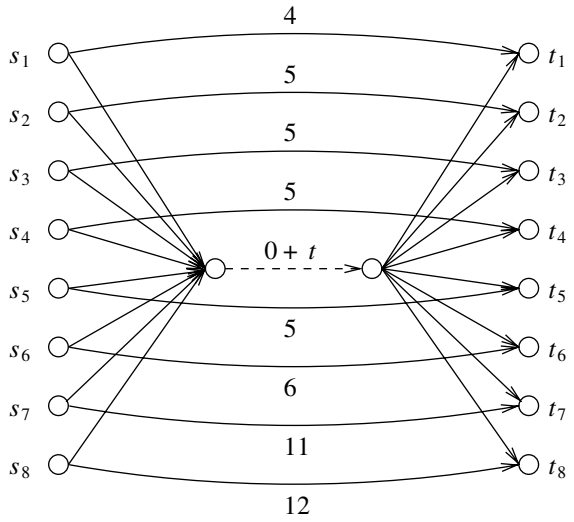

Fig. 1. An instance of MaxToll with 8 origin–destination pairs (s_k, t_k), $k = 1, \ldots, 8$, unitary demands and one toll arc (dashed). Arc labels refer nonzero fixed costs. Unlabeled arc have fixed cost 0. The optimal toll is $t = 5$ with a revenue of 35. All flows, except the one going from s_1 to t_1, go through the toll arc.

2.2. Complementarity constraints and smoothing

For a fixed toll vector t, the lower level of MaxToll is a linear program. If that linear program is replaced by its optimality conditions, MaxToll transforms into the single-level program:

$$(\mathrm{MT}) \quad \max_{t,z^k,\lambda^k,s^k} t'x$$

$$\forall k \in \mathscr{K} \begin{cases} Az^k = b^k, \\ A'\lambda^k + s^k = C(t), \\ Z^k s^k = 0, \qquad (*) \\ z^k, s^k \geqslant 0, \end{cases}$$

where 0 is the null vector of appropriate size, W is a diagonal matrix diag(w) whose entries match those of the vector w. Since a simple manipulation [22,23] can make the objective linear, it turns out that the main difficulty in solving (MT) is concentrated in the set of complementarity constraints (*). In addition to being nonlinear, it is well-known [6] that these constraints may fail to satisfy any constraint qualification, indicating that the problem is numerically ill-posed. In fact, these constraints are combinatorial constraints in disguise. One way to tackle them is to replace (*) by the equivalent[5] equation $\min(z^k, s^k) = 0$ (componentwise), and to smooth this new equation. Different such smoothing techniques have been proposed (see for e.g. [10,21,7,5,13]). One of them consists in solving, for a given positive value of μ, the parametric problem

[5] Together with z^k, $s^k \geqslant 0$.

$$Z^k s^k = \mu e,$$

$$z^k, \quad s^k \geqslant 0,$$

where $e = (1, 1, \ldots, 1)' \in \mathbb{R}^{m_x+m_y}$ and $\mu > 0$. This leads to the following mathematical program:

$$(\mathrm{MT}_\mu) \qquad \max_{t, z^k, \lambda^k, s^k} t'x$$

$$\forall k \in \mathscr{K} \left\{ \begin{array}{c} Az^k = b^k \\ A'\lambda^k + s^k = C(t) \\ Z^k s^k = \mu e \\ z^k, s^k \geqslant 0 \end{array} \right] (\mathrm{PD}_{t,\mu}),$$

where the parameter μ is driven to zero according to some sequence $\{\mu^l\}$. We will pursue this idea in Section 3. Note that the set of constraints in (MT_μ) is exactly the perturbed primal-dual system used in path-following interior-point methods for linear programming (see e.g. [4,35] for a gentle introduction). This ensures that for all $t \in \mathbb{R}^{m_x}$ and $\mu > 0$, there exists a unique solution $z^k_\mu(t) > 0$, $s^k_\mu(t) > 0$ and $\lambda^k_\mu(t)$ to the set of constraints—if a strictly interior point exists. Moreover, this solution is differentiable with respect to t and μ—given that A has full row rank. This leads to an implicit formulation of (MT_μ):

$$(\mathrm{IMT}_\mu) \quad \max_t t'x_\mu(t).$$

This smooth, unconstrained formulation is the starting point of our procedure.

In the view that bilevel programs (and MPECs) are generically intractable, solving (MT) looks problematic. Indeed, it does not satisfy most regularity assumptions used in the MPEC literature to prove convergence of standard algorithms to a stationary point. We briefly discuss three of them. One of the most stringent assumptions is the existence of a unique solution to the lower-level program when upper-level variables are fixed. In MaxToll, the lower level is a parametric linear program that admits a potentially infinite number of solutions for some values of the toll vector. Another strong assumption is the strict complementarity of the optimal solution, i.e., $z^k + s^k > 0$ for all k. This is usually not satisfied in (MT). Typically, the optimal solution consists in a set of path flows, one for each commodity. At least one of these paths is such that there exists an alternative path of identical length with the same origin/destination (otherwise tolls and revenue can be raised). This implies that the primal solution is not unique, and that strict complementarity does not hold.

In MPEC, the regularity condition MPEC-LICQ [29,31] relaxes the standard linear independence constraint qualification by requiring that the gradients of the active constraints, with the exception of the complementarity constraints, be linearly independent. In our situation, for every commodity k, a typical solution has at least one node with no flow from k passing through it. The corresponding constraints $z^k \geqslant 0$ are active and the gradients of these constraints suffice to construct, by linear combination, the gradient of the line of $Az^k = b^k$ corresponding to the node without flow. Hence, generically, MPEC-LICQ does not hold. However, we will see in the next sections that this lack of regularity does not impede on the good numerical behavior of the proposed method.

2.3. Exact solutions and inverse optimization

We first recall an exact formulation [22,23] that will be useful for comparisons. It is possible to obtain a global optimal solution to MaxToll by solving the standard mixed 0–1 program

$$(\text{MIP}) \qquad \max_{t,z^k,\lambda^k,s^k,t^k} \sum_{k\in\mathscr{K}} \sum_{i=1}^{m_x} n_k t_i^k$$

$$\forall k \in \mathscr{K} \begin{cases} Az^k = b^k, \\ A'\lambda^k \leqslant C(t), \\ C(0)'z^k + \sum_{i=1}^{m_x} t_i^k = b^{k\prime}\lambda^k, & (**) \\ -Mx_i^k \leqslant t_i^k \leqslant Mx_i^k \quad \forall i \in [m_x], & (***) \\ -M(1-x_i^k) \leqslant t_i^k - t_i \leqslant M(1-x_i^k) \quad \forall i \in [m_x], & (***) \\ z^k \geqslant 0, \\ x^k \in \{0,1\}^{m_x}, \end{cases}$$

where M is some large constant and $[n]$ denotes $\{1,\ldots,n\}$. In the above formulation, the complementarity constraints (*) have been replaced by the equivalent linearized constraint (**), using variables t_i^k that represent the revenue associated with arc i and commodity k. Equations (***) force the equality of commodity tolls on toll arcs that carry positive flows. The program (MIP) will be used to evaluate the quality of the solutions produced by our heuristic procedure.

Next, we will use the fact that one can efficiently find the best toll vector compatible with a lower-level flow solution. Indeed, assume that the paths used by the various commodities are known a priori. This implies that the variables z^k are fixed and that the complementarity constraints (*) become linear. Therefore, (MT) transforms into

$$(\text{MT}_z) \qquad \max_{t,\lambda^k,s^k} x't$$

$$\forall k \in \mathscr{K} \begin{cases} A'\lambda^k + s^k = C(t), \\ Z^k s^k = 0, \\ s^k \geqslant 0, \end{cases}$$

where $Z^k,\ k \in \mathscr{K}$ and x are fixed. This is a *linear program* that can be solved by computing shortest paths from all origins to all destinations (see [23]). This *inverse optimization* procedure will be used to optimize the toll vector with respect to the commodity flows produced by the smoothing algorithm.

3. A smoothing algorithm

3.1. The algorithm

The global smoothing heuristic (GSH), that is detailed in Fig. 2, is based on the implicit formulation (IMT_μ) presented in Section 2.2.

For given μ and t, the solution $x_\mu(t)$ is unique, and this allows to work in t-space only (at fixed μ). As a function of t, the objective $F_\mu(t) \equiv t'x_\mu(t)$ is nonlinear and has a large number of local optima (see Fig. 3). As μ increases, however, the smoothing tends to eliminate many of these optima. Our insight is that for μ large enough, $F_\mu(t)$ is almost unimodal and has a global optimum that corresponds roughly to the global optimum of $F_0(t)$. More precisely, one can slowly decrease μ toward 0, and follow the optimum of $F_\mu(t)$ to the global optimum of $F_0(t)$ (see Fig. 3). This is the idea behind the algorithm: GSH starts with a large μ, finds the optimum of $F_\mu(t)$ with a simple gradient method, decreases μ gradually, and recomputes the optimum at each step—starting from the previous solution—until μ is small enough. We claim that this

Algorithm. GLOBAL SMOOTHING HEURISTIC

Input: $\mathcal{K},\ C(\cdot),\ A,\ b^k,\ k \in \mathcal{K}$.

Smoothing parameters: $\overline{\mu} > \underline{\mu} > 0,\ 0 < \beta < 1$.

Tolerance: $\epsilon > 0$.

Step length: $\alpha > 0$.

Initial point: t^0, $(z^0,\ s^0,\ \lambda^0)$ solution to $(PD_{t^0,\overline{\mu}})$.

INITIALIZATION: Set $\mu \leftarrow \overline{\mu},\ (z,\ s,\ \lambda) \leftarrow (z^0,\ s^0,\ \lambda^0),\ t \leftarrow t^0,\ F \leftarrow t'x^0$.

MAIN ITERATION: **while** $\mu > \underline{\mu}$

repeat

– Compute the gradient, Δt, of $F_\mu(t) := t'x_\mu(t)$ at t.

– Set $F_{pr} \leftarrow F$ and $t := t + \alpha\Delta t$.

– Compute the solution $(z,\ s,\ \lambda)$ to $(PD_{t,\mu})$.

– Update the objective: $F \leftarrow F_\mu(t)$.

until $|F - F_{pr}| \le \epsilon$.

Set $\mu \leftarrow \beta\mu$ and compute the solution (z, s, λ) of $(PD_{t,\mu})$.

Output: toll vector t, flow vector z, revenue F.

Fig. 2. Algorithm global smoothing heuristic (GSH).

procedure yields near-optimal solutions. Although we have no theoretical result to back this claim, the numerical experiments reported in Section 4 support it.

3.2. Detailed description of GSH

Recall the definition of $(\mathrm{PD}_{t,\mu})$ in (MT_μ) of Section 2.2. To solve $(\mathrm{PD}_{t,\mu})$, we apply Newton's method. At each step, the linear systems

$$\left.\begin{aligned} A\Delta z^k &= r_z^k, \\ A'\Delta\lambda^k + \Delta s^k &= r_\lambda^k, \\ Z^k\Delta s^k + S^k\Delta z^k &= r_\mu^k \end{aligned}\right] (N^k)$$

are solved for $(\Delta z^k, \Delta\lambda^k, \Delta s^k)$, with the residuals

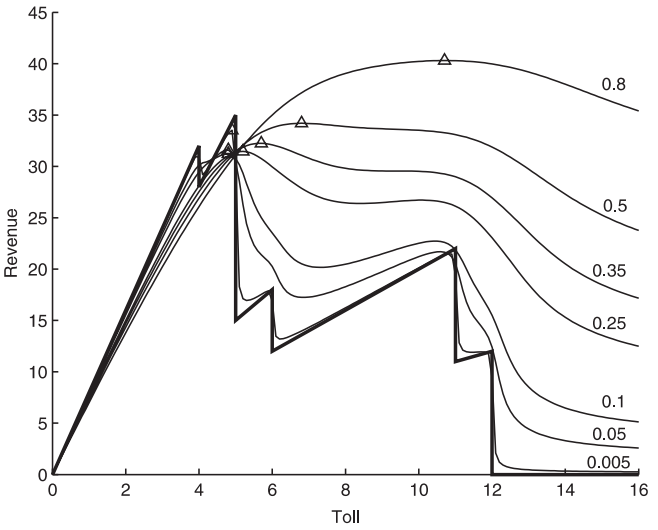

Fig. 3. Revenue vs. toll for different values of the smoothing parameter μ in the example of Fig. 1. Values of μ are indicated on the right next to each curve. The thick line corresponds to no smoothing and the upward triangles point to the maximum of each curve. By following these triangles, one is led to the global optimum of MaxToll.

$$r_z^k = b^k - Az^k,$$
$$r_\lambda^k = C(t) - A'\lambda^k - s^k,$$
$$r_\mu^k = \mu e - Z^k s^k.$$

This system reduces to the normal equations

$$[AZ^k(S^k)^{-1}A']\Delta\lambda^k = r_z + A(S^k)^{-1}(Z^k r_\lambda^k - r_\mu^k),$$
$$\Delta s^k = r_\lambda^k - A'\Delta\lambda^k,$$
$$\Delta z^k = -(S^k)^{-1}(-r_\mu^k + Z^k\Delta s^k).$$

If A has full row rank,[6] and if z^k and s^k are strictly positive, the matrix $[AZ^k(S^k)^{-1}A']$ is symmetric positive definite, and the system has a unique solution. We proceed with Newton steps until

$$\frac{\|r_z^k\|}{\max(1,\|b^k\|)} + \frac{\|r_\lambda^k\|}{\max(1,\|\mathscr{C}(t)\|)} + \frac{\|r_\mu^k\|}{\max(1,\|\mu e\|)} < \epsilon',$$

where $\|\cdot\|$ denotes the Euclidean norm and ϵ' is a strictly positive tolerance.

Another major step of GSH is the computation of the gradient of $F_\mu(t) = t'x_\mu(t)$. This is given by the implicit function theorem. Let D_t be the differential operator with respect to t. Since one requires $D_t z^k$ to derive

$$D_t F_\mu(t) = x_\mu(t) + (t' D_t x_\mu(t))',$$

[6] We assume that the underlying undirected graph is connected so that A has full row rank when an arbitrary row is removed.

one must solve the systems N^k, for all k, with the residuals r_z^k, r_λ^k, r_μ^k replaced by

$$r_z^k = 0,$$
$$r_\lambda^k = \begin{pmatrix} e_i \\ 0 \end{pmatrix},$$
$$r_\mu^k = 0,$$

where e_i is the unit vector of size m_x in the ith component direction.

One last point that we need to address is the choice of α, the gradient step length. Because the very computation of the objective $F_\mu(t)$ requires solving $(\mathrm{PD}_{t,\mu})$ repeatedly, performing a line search is out of the question. In Fig. 2, we actually set the stepsize to a constant value. Since a constant α performs very poorly and may even prevent convergence, we opted for a compromise that exploits the bilinearity of the objective function. In the direction of Δt, $F_\mu(t)$ can be approximated in the following fashion:

$$F_\mu(t+\alpha\Delta t) = (t+\alpha\Delta t)' x_\mu(t+\alpha\Delta t) \approx (t+\alpha\Delta t)'(x_\mu(t) + \alpha(\Delta t' D_t x_\mu(t))').$$

This last expression is maximized at

$$\alpha^* = -\frac{1}{2}\frac{(D_t F_\mu(t))'(D_t F_\mu(t))}{(D_t F_\mu(t))' D_t x_\mu(t)(D_t F_\mu(t))}$$

and this value was adopted in our numerical experiments (the algorithm in Fig. 2 should be modified accordingly). Note that we sidestep the computation of the second derivative, which is extremely time-consuming.

3.2.1. Pre- and post-processing

Solving $(\mathrm{PD}_{t,\mu})$ requires the existence of a strictly interior solution to $(\mathrm{PD}_{t,0})$. In MaxToll, a strictly feasible primal solution for a given commodity k is a flow with source s^k and sink t^k using all arcs in the network; this must be satisfied for each commodity separately. Such a flow exists if and only if for each arc a and each commodity k, there is a path from s^k to t^k that includes a. (*Note*: this is not satisfied in the example of Fig. 1.) One way of achieving this is to add one node that is linked to every other node (both ways) with toll-free arcs of arbitrarily high fixed cost; the drawback of this approach is that many arcs need to be added. Typically, however, each arc connects both the source and the sink of at least one commodity (otherwise this arc is useless and can be removed). Therefore, one only needs to connect the origins and destinations to an extra node. This is the solution adopted in the numerical tests. Once the existence of a strictly feasible primal solution is guaranteed, the boundedness of the primal feasible set ($0 \leqslant z^k \leqslant n_k$) ensures that a strictly feasible dual solution exists as well (see e.g. [28]).

Algorithm GSH returns a flow and a toll vector. Since the lower threshold $\underline{\mu}$ is not 0 and should not be too small in order to improve computing time and prevent numerical instability,[7] the solution returned is only approximate. The following post-processing step fixes the problem. For each commodity and starting from the corresponding origin, we greedily construct a path by selecting the arcs having the largest commodity flow, until the destination is reached. Since all initial costs are nonnegative, there are no negative cycles and the procedure is bound to terminate finitely. Then, we use the inverse optimization step described in Section 2.3 to compute the best tolls consistent with the selected paths.[8]

[7] When μ is small, $D_t z_\mu^k(t)$ becomes nearly singular because of the absence of strict complementarity.

[8] Note that the converse approach that consists in keeping the tolls provided by GSH and computing the best paths for these tolls makes less sense because of the discontinuities in the revenue function (see Fig. 3). A priori, one does not know on which side of the discontinuity GSH will land.

4. Numerical results

The numerical tests presented in this section give a fairly good idea of the performance of the algorithm and suggest possible directions of improvement. We proceed in three steps. First we choose appropriate values for the parameters of the heuristic. Then, on a set of instances of varying sizes, we estimate the performance of the algorithm by comparing the output to *the* exact solution. Finally we test the algorithm on a family of "hard" instances. The tests were performed on a Sun Ultra 60 workstation. The exact algorithm uses the standard MIP solver of CPLEX 8.1 with the formulation (MIP). Algorithm GSH runs on MATLAB 6.0.

4.1. Problem instances

Pure random networks tend to be fairly easy to solve for the exact algorithm and were consequently dismissed. Instead, we opted for a more structured family of instances that proved to be hard to solve exactly for moderately large sizes. The underlying network is a Manhattan-like grid illustrated in Fig. 4. Our problems involve unit demand for every commodity. Each arc has a fixed cost uniformly chosen in the range $\{2, 3, \ldots, 20\}$, and its probability of being a toll arc is set to p. Sources and sinks are located arbitrarily on the "boundary" of the network. To ensure that there exists a toll-free path between each origin–destination (OD) pair, we add one toll-free arc of arbitrarily large cost between each pair. In the tests, we used square grids of size 5×5 to 10×10 with density p varying between 10% and 50%.

4.2. Choice of parameters

Preliminary experiments indicate that the parameters $\bar{\mu}$, $\underline{\mu}$ and β strongly influence the performance of the algorithm. Other parameters play a lesser role and were set to the following arbitrary values:

$$\epsilon = 0.025,$$
$$\epsilon' = 0.1.$$

The parameter α is set to α^* as explained in Section 3.2.

Parameter $\bar{\mu}$ seems to be the most critical. To set its value, we fixed β and $\underline{\mu}$ to conservative values, $\beta = 0.9$ and $\underline{\mu} = 0.001$, respectively, and tested the algorithm with different values of $\bar{\mu}$ on networks of size

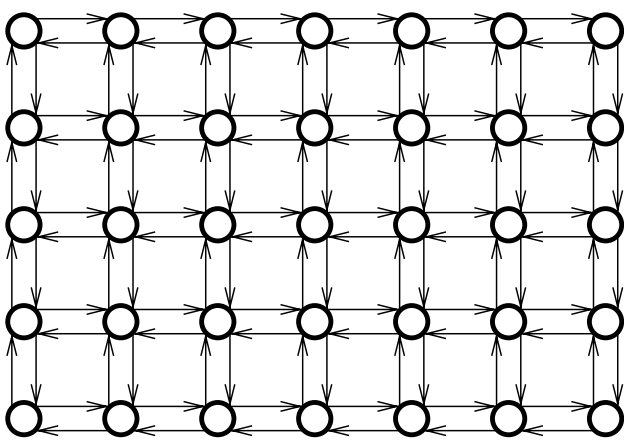

Fig. 4. The underlying network (size may vary).

5×5, with $p = 10\%$ and 6 ODs. The corresponding results are shown in Fig. 5, where each point is the average of 5 random instances. The value $\bar{\mu} = 0.20$ gives the best results. To determine the dependency of the appropriate value of $\bar{\mu}$ on the size of the instance, we performed identical tests on networks of size 10×10 with $p = 10\%$ and 16 origin–destination pairs, as well as on networks of size 5×5 with $p = 20\%$ and 6 OD pairs. The results are shown in Fig. 5. Again, $\bar{\mu} = 0.20$ gives the best results. It is striking that the quality of the solution decreases as the parameter $\bar{\mu}$ increases, although this is not a surprising result. Indeed, we expect poor solutions for small $\bar{\mu}$, as the smoothing is not strong enough to prevent the algorithm from being trapped in local optima. However, performance also decreases for large values of $\bar{\mu}$. This

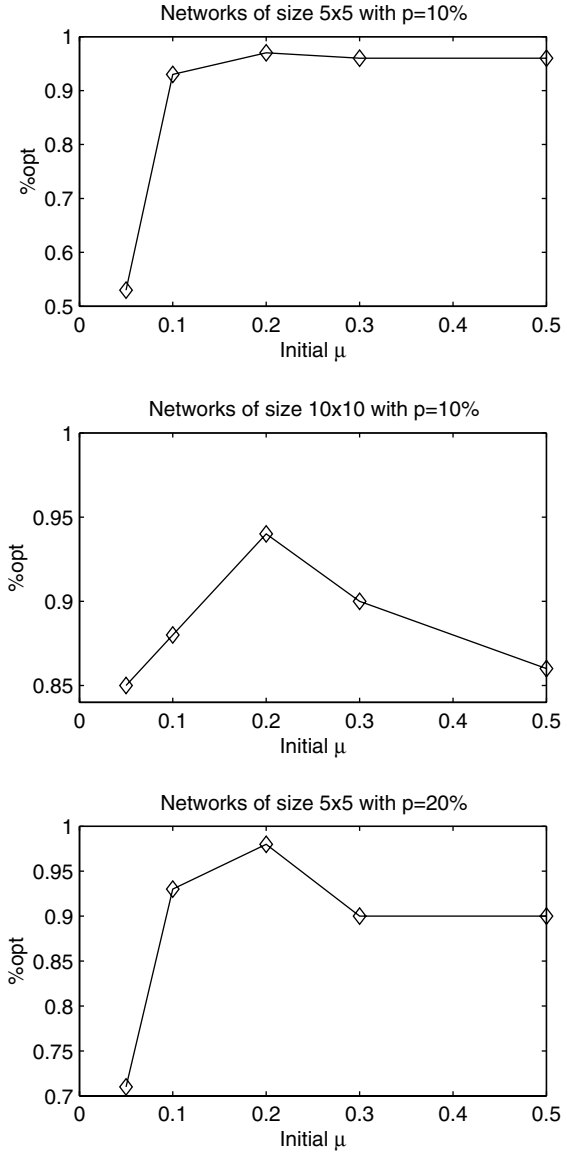

Fig. 5. Accuracy for varying values of $\bar{\mu}$ (referred to as initial μ).

probably comes from the fact that in the initial stages of the algorithm, a rough smoothing takes the algorithm very far from the optimal tolls. The algorithm then gets stuck into the flat valley in the large tolls region (see Fig. 3). A larger value of β and a smaller value of ϵ may help to recover properly from that situation.

We proceeded similarly for the parameters β and $\underline{\mu}$. For β, we set $\bar{\mu} = 0.2$ and $\underline{\mu} = 0.001$ and tested the heuristic on the same networks as for $\bar{\mu}$. The results are shown in Fig. 6. In this case, the quality of the results depend on problem size, which led to our choosing a cautious value of $\beta = 0.9$. The figures suggest

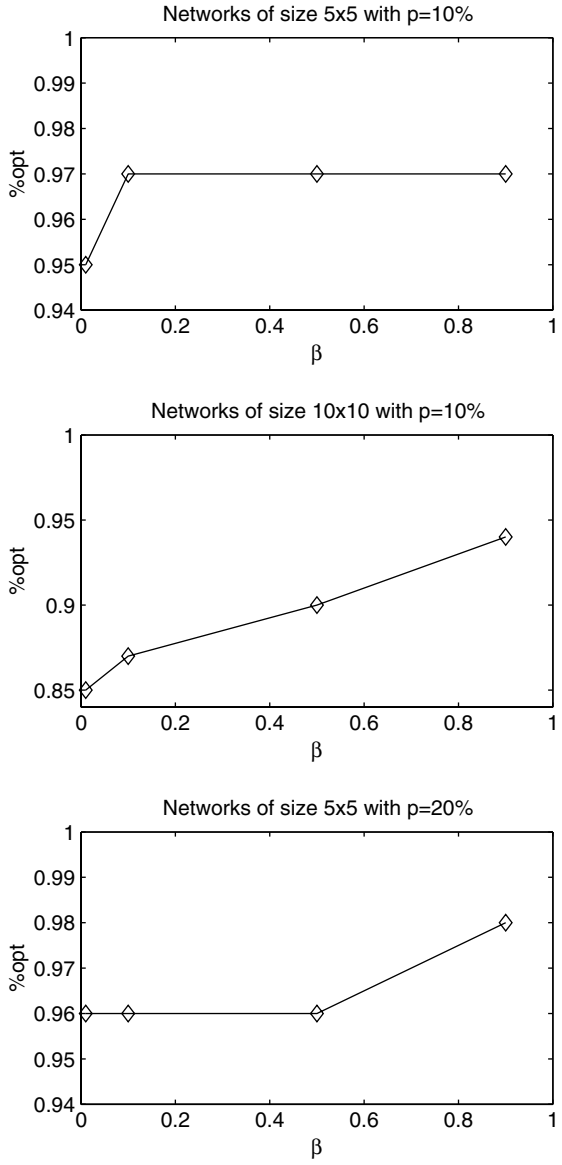

Fig. 6. Accuracy for varying values of β.

that an even larger value of β might give better results. However, this advantage is somewhat offset by the large computation times associated with large values of β.

For $\underline{\mu}$, we set $\bar{\mu} = 0.2$ and $\beta = 0.9$ and tested GSH on the same instances. The results are shown in Fig. 7. Similar to the parameter β, the quality of the results depends on problem size. We chose $\underline{\mu} = 0.01$. For the rest of the numerical tests, the values of these parameters are fixed. Note finally that, contrary to the case of $\bar{\mu}$, the relationship between the quality of the solution and the value of β and $\underline{\mu}$ is in line with our intuition: quality is an increasing function of β (respectively a decreasing function of $\underline{\mu}$).

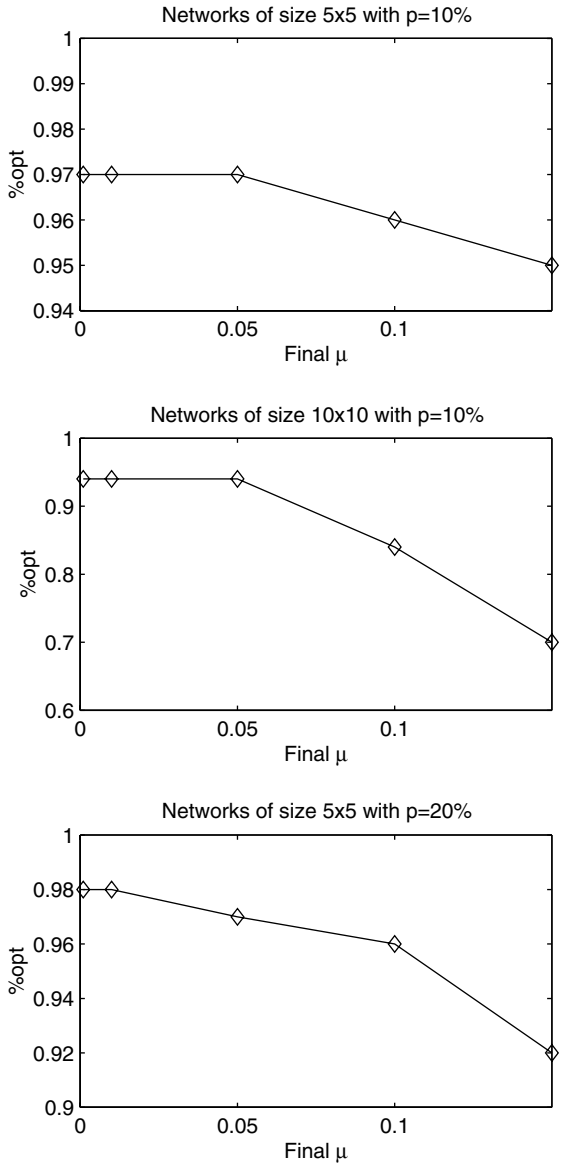

Fig. 7. Accuracy for varying values of $\underline{\mu}$ (refered to as final μ).

Table 1
Accuracy and computation time for problems with varying sizes

	Instance					
	5 × 5	6 × 6	7 × 7	8 × 8	9 × 9	10 × 10
n	25	36	49	64	81	100
m	86	128	178	236	302	376
K	6	8	10	12	14	16
p	10%	10%	10%	10%	10%	10%
%OPT	97.5 (6.6)	97.1 (3.4)	94.5 (4.5)	95.6 (4.0)	93.3 (5.1)	95.1 (3.8)
Min.	77.8	89.7	87.0	87.2	82.4	88.1
Max.	100	100	100	100	100	100
%LP	87.3	84.8	77.5	80.9	78.9	82.6
t_{GSH}	29.6 (3.1)	58.1 (7.3)	100.2 (10.4)	166.1 (11.4)	250.2 (24.1)	385.3 (30.4)
t_{CPLEX}	0.6 (0.1)	1.1 (0.6)	1.4 (0.5)	4.9 (3.8)	24.6 (34.1)	9.7 (5.8)

4.3. Solution accuracy

A natural measure of the quality of a suboptimal solution is the ratio of its objective with respect to the global optimum. Expressed as a percentage, we call this ratio the accuracy of the solution. To estimate the accuracy of the solutions produced by GSH, we performed three series of tests. In the first series, we used problems of increasing size. The results are shown in Table 1. Each column provides averages over 10 random instances.[9] Throughout this section and the next, we use the following notation:

- n: number of nodes,
- m: number of arcs,
- K: number of OD pairs,
- p: average fraction of toll arcs,
- %OPT: ratio of the GSH solution over the global optimum,
- %LP: objective value returned by heuristic divided by linear relaxation value,
- t_{GSH}: computation time of GSH (in seconds),
- t_{CPLEX}: computation time of exact algorithm (in seconds).

Numbers in parentheses indicate the standard deviation, expressed in the same units as the averages.

From the results of Table 1, we observe that the accuracy of GSH lies around 95%, and that the corresponding standard deviation is small. While smaller instances are easier to solve, the performance is not much influenced by problem size. Accuracy can be as high as 100% and as low as 75%. However, a quick experimentation suggested that low accuracy cases can be significantly improved through a suitable choice of parameters $\bar{\mu}$ and β.

The computation time of algorithm GSH is roughly linear in $m \cdot K$. This is what we expect: in the absence of capacity constraints, the implicit formulation decouples the commodity flows. By exploiting the network structure, the linear systems can be solved in a time linear in m for each OD pair. It is hard to deduce anything from the values of t_{CPLEX} except that it seems to grow slightly faster than t_{GSH}. It is clear though that the exact algorithm runs faster on the small instances. Note however that our ultimate goal is to demonstrate that the heuristic performs better on large instances, and this will be done in the next section. The objective of this section is only to get reliable estimates for the accuracy of the heuristic. Also note that

[9] To avoid degenerate instances, we considered only those instances with an optimal objective value of at least 10.

Table 2
Accuracy and computation time for problems with varying fractions of toll arcs

	Instance				
	7×7 $p = 10\%$	7×7 $p = 20\%$	7×7 $p = 30\%$	7×7 $p = 40\%$	7×7 $p = 50\%$
n	49	49	49	49	49
m	178	178	178	178	178
K	10	10	10	10	10
p	10%	20%	30%	40%	50%
%OPT	94.3 (3.7)	96.4 (4.4)	92.6 (9.0)	95.5 (4.8)	95.5 (2.8)
Min.	89.1	87.0	74.5	85.3	89.9
Max.	100	100	100	100	98.8
%LP	78.3	83.6	83.9	87.6	92
t_{GSH}	101.0 (9.9)	116.1 (11.4)	147.3 (9.6)	178.7 (13.8)	204.1 (13.8)
t_{CPLEX}	1.5 (0.5)	2.4 (1.9)	40.9 (46.5)	62.6 (87.0)	1974.8 (4851.1)

a rough implementation of the heuristic using MATLAB hardly compares to a state-of-the-art mathematical programming software like CPLEX.

The second series of tests aims at determining the impact of the number of toll arcs on the performance of the algorithm. For this, we used networks of size 7×7 with 10 OD pairs and different values of p, the average fraction of toll arcs. Results are reported in Table 2. Again, the accuracy is roughly 95% and does not depend much on the value of p. High and low values are roughly the same. What is interesting here is the computation time. For the heuristic algorithm, t_{GSH} grows sublinearly in p while the growth of the computing time of the exact method, t_{CPLEX}, is clearly exponential in p, because of the presence of one binary variable for each toll arc. Actually, we were not able to obtain results for values of p larger than 50%, due to running time limits set to 60 min.

The third series of tests focuses on the impact of the number of OD pairs. We used networks of size 7×7 with $p = 50\%$ and increasing values of K. The results are given in Table 3. Here the accuracy of GSH improves significantly. In particular, for the case of a single OD pair, the algorithm almost reaches a perfect score, with the optimal value obtained in all but one instance. The computation time t_{GSH} is roughly linear in K as expected[10] and t_{CPLEX} grows exponentially in K due to the fact that the number of binary variables in (MIP) is proportional to the number of OD pairs.

4.4. Hard instances

Next, we consider a set of instances for which the exact algorithm was unable to guarantee an optimal solution within a reasonable time frame. Since the optimal value was unavailable, we used the linear relaxation as a bound on the accuracy of GSH. However, extrapolating the results of the preceding section, we still expect the accuracy of GSH to be of the order of 95%.

As computation time t_{CPLEX} of the algorithm is exponential in p and K, we constructed moderately large instances with high values of these two parameters. More precisely, we designed networks of size 10×10 with $p = 50\%$ and $K = 16$. The results of the tests are detailed in Table 4. The linear programming upper bound indicates that the average accuracy of GSH is at least 90%. In all cases the exact algorithm was unable to find an optimal solution after t_{GSH} seconds (after what it was stopped). In all but one case, GSH found a better solution than the best integer solution found by CPLEX. In two cases, CPLEX found no

[10] The computation time reported in the table is not exactly linear in $m \cdot K$. This is particularly obvious for the last columns, due to the post-processing stage which uses MATLAB's linear programming algorithm. If we had used CPLEX instead for that last stage of the heuristic, we would have gained as much as 75% in computation time and t_{GSH} would have been linear in $m \cdot K$.

Table 3
Accuracy and computation time for problems with varying numbers of OD pairs

	Instance					
	7×7 $K = 1$	7×7 $K = 2$	7×7 $K = 4$	7×7 $K = 6$	7×7 $K = 8$	7×7 $K = 10$
n	49	49	49	49	49	49
m	178	178	178	178	178	178
K	1	2	4	6	8	10
p	50%	50%	50%	50%	50%	50%
%OPT	99.3 (2.1)	98.0 (1.7)	97.1 (2.4)	95.1 (4.5)	97.2 (1.8)	95.5 (2.8)
Min.	92.9	95.0	91.0	85.2	94.1	89.9
Max.	100	100	100	99.4	100	98.8
%LP	99.3	95.3	95.3	91.2	94.0	92.2
t_{GSH}	11.3 (0.7)	25.4 (1.4)	55.4 (3.1)	90.0 (3.0)	138.9 (6.7)	204.1 (13.8)
t_{CPLEX}	0.6 (0.03)	0.8 (0.2)	1.8 (0.9)	36.6 (64.0)	61.0 (77.2)	1974.8 (4851.1)

Table 4
Results on "hard" instances with $n = 100$, $m = 376$, $K = 16$ and $p = 50\%$

Instance	%LP		t_{GSH}	t^*_{GSH}
	GSH	CPLEX		
1	89.3	0	4095.0	511.3
2	89.9	88.3	4195.8	487.2
3	91.8	89.6	4912.4	514.6
4	90.8	89.7	4675.2	510.1
5	96.4	94.3	2648.3	511.05
6	93.6	92.3	4139.9	518.53
7	93.5	93.4	4325.8	530.33
8	86.1	97.9	4258.5	495.27
9	71.4	0	3833.3	479.97
10	93.0	89.0	3885.0	531.81
Average	89.9 (7.0)	73.5 (38.8)	4096.9 (605.8)	509.0 (16.2)

In all cases, the exact algorithm was stopped after t_{GSH} seconds and the optimal solution was unknown. In the CPLEX column, we used the best integer solution found after t_{GSH} seconds. t^*_{GSH} is the computation time without post-processing.

integer solution before reaching the preset time limit. Although the values of t_{GSH} are quite high, more than 75% of the computing time is taken up by the post-processing step. As noted previously, the LP solver of CPLEX would have solved this structured LP, actually a transportation problem, almost instantaneously. A better estimate of the "optimal" computation time of the algorithm is obtained by subtracting this value (see last column of Table 4).

5. Concluding remarks

In this paper, we designed an algorithm for solving a class of structured bilevel programs that arises naturally in pricing applications. While the smoothing approach has been proposed previously in the MPEC literature, our implementation is mainly concerned with semi-global optimality. The performance of the algorithm was tested on network pricing problems of significant sizes, and its performance proved very good to excellent, given the difficulty of the problem instances. More significant, the running time grows

linearly with problem size. This resulted in an algorithm consistently outperforming a MIP solver on "hard" instances, despite our cursory implementation.

One advantage of our approach is that it can be applied to a class of problems much larger than the one considered in this paper. Our choice of network pricing problems was mainly motivated by the availability of a software for finding a globally optimal solution, which was instrumental in assessing the quality of the method. However, Algorithm GSH applies to more general models, including network-free ones. Our computational results show that the smoothing technique underlying the algorithm is a powerful tool that could extend to other bilevel environments.

References

[1] C. Audet, P. Hansen, B. Jaumard, G. Savard, Links between linear bilevel and mixed 0–1 programming problems, Journal of Optimization Theory and Applications 93 (1997) 273–300.
[2] J. Bard, J. Plummer, J. Sourie, A bilevel programming approach for determining tax credits for biofuel production, European Journal of Operational Research 120 (2000) 30–46.
[3] H. Benson, D.F. Shanno, R. Vanderbei, Interior-point methods for nonconvex nonlinear programming: Complementarity constraints. Technical Report ORFE-02-02, Operations Research and Financial Engineering, Princeton University, 2002.
[4] D. Bertsimas, J. Tsitsiklis, Introduction to Linear Optimization, Athena Scientific, Belmont, MA, 1997.
[5] Billups, S.C., K. Murty, Complementarity problems, Journal of Computational and Applied Mathematics 124 (2000) 303–318.
[6] Y. Chen, M. Florian, The nonlinear bilevel programming problem: Formulations, regularity and optimality conditions, Optimization 32 (1995) 193–209.
[7] C. Chen, O. Mangasarian, A class of smoothing functions for nonlinear and mixed complementarity problems, Computational Optimization and Applications 5 (1996) 97–138.
[8] F. Clarke, Optimization and Nonsmooth Analysis, Wiley, New York, 1983.
[9] J.P. Côté, P. Marcotte, G. Savard, A bilevel modelling approach to pricing and fare optimization in the airline industry, Journal of Revenue and Pricing Management 2 (2003) 23–36.
[10] S. Dempe, Annotated bibliography on bilevel programming and mathematical programs with equilibrium constraints, Optimization 52 (2003) 333–359.
[11] S. Dempe, J. Bard, A bundle trust-region algorithm for bilinear bilevel programming, Journal of Optimization Theory and Applications 110 (2001) 265–288.
[12] F. Facchinei, H. Jiang, L. Qi, A smoothing method for mathematical programs with equilibrium constraints, Mathematical Programming 85 (1999) 107–134.
[13] M.C. Ferris, C. Kanzow, Complementarity and related problems: A survey, in: P.M. Pardalos, M.G.C. Resende (Eds.), Handbook of Applied Optimization, Oxford University Press, New York, 2002, pp. 514–530.
[14] R. Fletcher, S. Leyffer, Numerical experience with solving MPECs as NLPs. Report NA-210, University of Dundee, 2002.
[15] R. Fletcher, S. Leyffer, D. Ralph, S. Scholtes, Local convergence of SQP methods for mathematical programs with equilibrium constraints. Report NA-209, University of Dundee, 2002.
[16] M. Fukushima, Z.Q. Luo, J. Pang, A globally convergent sequential quadratic programming algorithm for mathematical programs with linear complementarity constraints, Computational Optimization and Applications 10 (1998) 5–34.
[17] M. Fukushima, J. Pang, Convergence of smoothing continuation method for mathematical programs with equilibrium constraints, in: M. Théra, R. Tichatschke (Eds.), Ill-posed Variational Problems and Regularization Techniques, Springer-Verlag, New York, 1999, pp. 99–110.
[18] M.R. Garey, D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman, New York, 1979.
[19] R. Jeroslow, A polynomial hierarchy and a simple model for competitive analysis, Mathematical Programming 32 (1985) 146–164.
[20] H. Jiang, D. Ralph, Smooth SQP methods for mathematical programs with nonlinear complementarity constraints, SIAM Journal of Optimization 10 (2000) 779–808.
[21] C. Kanzow, Some noninterior continuation methods for linear complementarity problems, SIAM Journal on Matrix Analysis and Applications 17 (1996) 851–868.
[22] M. Labbé, P. Marcotte, G. Savard, A bilevel model of taxation and its application to optimal highway pricing, Management Science 44 (1998) 1595–1607.
[23] M. Labbé, P. Marcotte, G. Savard, On a class of bilevel programs, in: Di Pillo, Giannessi (Eds.), Nonlinear Optimization and Related Topics, Kluwer Academic Publishers, 1999, pp. 183–206.
[24] S. Leyffer, The Penalty Interior Point Method fails to converge for mathematical programs with equilibrium constraints, University of Dundee Report NA-208, 2002.

[25] Z.-Q. Luo, J.-S. Pang, D. Ralph, Mathematical Programming with Equilibrium Constraints, Cambridge University Press, UK, 1996.
[26] J. Outrata, M. Kocvara, J. Zowe, Nonsmooth Approach to Optimization Problems with Equilibrium Constraints: Theory, Applications and Numerical Results, Kluwer Academic Publishers, Dordrecht, Netherlands, 1998.
[27] S. Roch, G. Savard, P. Marcotte, An approximation algorithm for Stackelberg network pricing, Networks 46 (2005) 57–67.
[28] C. Roos, T. Terlaky, J.-Ph. Vial, Theory and Algorithms for Linear Optimization: An Interior Point Approach, Wiley, Chichester, UK, 1997.
[29] H. Scheel, S. Scholtes, Mathematical programs with complementarity constraints: Stationarity, optimality and sensitivity, Mathematics of Operations Research 25 (2000) 1–22.
[30] S. Scholtes, Convergence properties of a regularization scheme for mathematical programs with equilibrium constraints, SIAM Journal of Optimization 11 (2001) 918–936.
[31] S. Scholtes, M. Stöhr, How stringent is the linear independence assumption for mathematical programs with complementarity constraints? Mathematics of Operations Research 26 (2001) 851–863.
[32] K. Shimizu, Y. Ishizuka, J. Bard, Nondifferentiable and Two-Level Mathematical Programming, Kluwer, Boston, 1997.
[33] L.N. Vicente, P. Calamai, Bilevel and multilevel programming: A bibliography review, Journal of Global Optimization 5 (1994) 291–306.
[34] L. Vicente, G. Savard, J. Júdice, Descent approaches for quadratic bilevel programming, Journal of Optimization Theory and Applications 81 (1994) 379–399.
[35] S. Wright, Primal-Dual Interior-Point Methods, SIAM, Philadelphia, PA, 1997.