Intersection cuts for bilevel optimization

Matteo Fischetti [*1], Ivana Ljubić [†2], Michele Monaci [‡3], and Markus Sinnl[§4]

[1] *DEI, University of Padua, Italy.*
[2] *ESSEC Business School of Paris, France.*
[2] *DEI, University of Bologna, Italy.*
[4] *ISOR, University of Vienna, Austria.*

Abstract

We address a generic Mixed-Integer Bilevel Linear Program (MIBLP), i.e., a bilevel optimization problem where all objective functions and constraints are linear, and some/all variables are required to take integer values. Rather than proposing an ad-hoc method applicable only to specific cases, we describe a general-purpose MIBLP approach.

We first propose necessary modifications needed to turn a standard branch-and-bound MILP solver into an exact and convergent MIBLP solver, also addressing MIBLP unboundedness and infeasibility. Contrarily to other approaches, in our convergent framework both leader and follower problems can be of mixed-integer type—provided that the leader variables influencing follower's decisions are integer and bounded.

We then introduce new classes of linear inequalities to be embedded in this branch-and-cut framework, some of which are intersection cuts based on feasible-free convex sets. We present a computational study on various classes of benchmark instances available from the literature, in which we demonstrate that our approach outperforms alternative state-of-the-art MIBLP methods.

1 Introduction

Bilevel optimization is a very challenging topic that received much of attention in recent years, as witnessed by the flourishing recent literature. In this paper we address a generic Mixed-Integer Bilevel Linear Program (MIBLP), i.e., a bilevel optimization problem where all objective functions and constraints are linear, and some/all variables are required to take integer values.

We aim at developing a general-purpose exact algorithm applicable to a general MIBLP (under mild conditions), rather than ad-hoc methods for specific cases. Our algorithmic design choice was to look for non-invasive supplements needed to convert an effective branch-and-cut Mixed-Integer Linear Programming (MILP) exact code into a valid MIBLP solver. In this way, we inherit the wide arsenal of tools (cuts, heuristics, propagations, etc.) available in modern MILP solvers, and do not have to address non-bilevel specific issues like numerical stability, effective LP parametrization, multi-threading support, and alike. This is a distinguished feature of our work: instead of proposing sophisticated bilevel-specific solution schemes for MIBLP, as in most papers from the literature, we build on a stable and powerful MILP solver and add bilevel-specific features to it.

We first introduce necessary modifications of branching, node-evaluation and fathoming rules of a standard branch-and-bound based MILP solver, and prove that they lead to an exact and (under some mild

[*]matteo.fischetti@unipd.it
[†]ivana.ljubic@essec.edu
[‡]michele.monaci@unibo.it
[§]markus.sinnl@univie.ac.at

conditions) convergent MIBLP solver. We then propose the use of Intersection Cuts (IC's) [3] within this branch-and-cut framework. Our approach relies on defining appropriate convex feasible-free sets that can be used to cut off bilevel infeasible points obtained from a problem relaxation. As far as we know, this is the first time IC's have been exploited in the context of bilevel programming. Many of the generic MIBLP approaches proposed in the literature are illustrated using small numerical examples involving few decision variables only (see, e.g. [2, 11, 15, 18]). To our knowledge, the present paper gives one of the most extensive computational studies for general MIBLP solvers ever reported in the literature. Extensive computational results on a testbed containing more than 300 instances from the literature with up to 80,000 variables are presented. An outcome of our experiments is that our approach outperforms available state-of-the-art MIBP solvers by a large margin.

The paper is organized as follows. Section 2 gives a brief introduction to general bilevel optimization and to MIBLP. Section 3 presents a basic MILP-based branch-and-bound algorithm for MIBLP, and proves its finite convergence under appropriate assumptions. An improved branch-and-cut algorithm is then described in Section 4, where we present two new families of valid MIBLP cuts along with their separation procedures. In Section 5, results of our computational study are provided. Some conclusions and possible directions for future work are finally addressed in Section 6.

A very preliminary version of our work appeared in [10].

2 The problem

A general bilevel optimization problem is defined as

$$\min_{x\in\mathbb{R}^{n_1}, y\in\mathbb{R}^{n_2}} F(x,y) \tag{1}$$

$$G(x,y) \leq 0 \tag{2}$$

$$y \in \arg\min_{y'\in\mathbb{R}^{n_2}} \{f(x,y') : g(x,y') \leq 0\}, \tag{3}$$

where $F, f : \mathbb{R}^{n_1+n_2} \to \mathbb{R}$, $G : \mathbb{R}^{n_1+n_2} \to \mathbb{R}^{m_1}$, and $g : \mathbb{R}^{n_1+n_2} \to \mathbb{R}^{m_2}$. Let $n = n_1 + n_2$ denote the total number of decision variables, and let $N_x = \{1, \ldots, n_1\}$ and $N_y = \{1, \ldots, n_2\}$ denote the index sets of the x and y variables, respectively.

We will refer to $F(x,y)$ and $G(x,y) \leq 0$ as the *leader* objective function and constraints, respectively, and to (3) as the *follower* subproblem. In case the follower subproblem has multiple optimal solutions, we assume that one with minimum leader cost among those with $G(x,y) \leq 0$ is chosen—i.e. we consider the *optimistic* version of bilevel optimization. The reader is referred to [14] for a comprehensive discussion on this topic.

By defining the follower value function for a given $x \in \mathbb{R}^{n_1}$

$$\Phi(x) = \min_{y\in\mathbb{R}^{n_2}} \{f(x,y) : g(x,y) \leq 0\}, \tag{4}$$

one can restate the bilevel optimization problem as follows:

$$\min F(x,y) \tag{5}$$

$$G(x,y) \leq 0 \tag{6}$$

$$g(x,y) \leq 0 \tag{7}$$

$$(x,y) \in \mathbb{R}^n \tag{8}$$

$$f(x,y) \leq \Phi(x). \tag{9}$$

Note that the above optimization problem would be hard (both theoretically and in practice) even if one would assume convexity of F, G, f and g (which would imply that of Φ), due to the intrinsic nonconvexity of (9).

Dropping condition (9) leads to the so-called *High Point Relaxation* (HPR). Clearly, if HPR is infeasible, so is its bilevel counterpart. Thus, without loss of generality, we will assume HPR is feasible.

As HPR contains all the follower constraints, any HPR solution (x, y) satisfies $f(x, y) \geq \Phi(x)$, hence (9) actually implies $f(x, y) = \Phi(x)$. Notice that replacing $\Phi(x)$ with some more tractable underestimator allows one to derive a heuristic solution, i.e., an upper bound on the value of optimal bilevel solution. Similarly, the solution of the problem obtained overestimating $\Phi(x)$ provides a valid lower bound for the problem. An HPR solution (x, y) will be called *bilevel infeasible* if it violates (9). A point $(x, y) \in \mathbb{R}^n$ will be called *bilevel feasible* if it satisfies all constraints (6)–(9).

2.1 Mixed-Integer Bilevel Linear Program

In the remaining part of the paper we will focus on the relevant special case in which some/all variables are required to be integer, and all leader/follower constraints and objective functions are linear/affine functions. To be more specific, we will consider the following Mixed-Integer Bilevel Linear Program (MIBLP), in its value-function reformulation (5)-(9):

$$\min c_x^T x + c_y^T y \tag{10}$$
$$G_x x + G_y y \leq q \tag{11}$$
$$Ax + By \leq b \tag{12}$$
$$x^- \leq x \leq x^+ \tag{13}$$
$$y^- \leq y \leq y^+ \tag{14}$$
$$x_j \text{ integer}, \quad \forall j \in J_x \tag{15}$$
$$y_j \text{ integer}, \quad \forall j \in J_y^L \tag{16}$$
$$d^T y \leq \Phi(x) \tag{17}$$

where c_x, c_y, G_x, G_y, q, A, B, b, x^-, x^+, y^-, and y^+ are given rational matrices/vectors of appropriate size, while sets $J_x \subseteq N_x$ and $J_y^L \subseteq N_y$ (where superscript L stands for leader) identify the (possibly empty) indices of the integer-constrained variables in x and y, respectively. Constraints (13)-(14) define explicit lower/upper bounds on the variables; as customary, we allow some entries in x^-, x^+, y^-, y^+ to be $\pm\infty$.

As to the value function $\Phi(x)$ for a given x, it is computed by solving the *follower MILP*:

$$\Phi(x) := \min d^T y \tag{18}$$
$$Ax + By \leq b \tag{19}$$
$$l \leq y \leq u \tag{20}$$
$$y_j \text{ integer}, \quad \forall j \in J_y \tag{21}$$

where l and u are, respectively, lower and upper bounds on the y variables in the follower—possibly $l_j = -\infty$ and/or $u_j = +\infty$ for some j. Without loss of generality, one can assume $l \leq y^-$ and $u \geq y^+$, i.e., the bound constraints (14) on the y variables are not weaker than the corresponding bounds (20) in the follower problem. For the same reason, one can assume that $J_y \subseteq J_y^L$.

Note that the follower objective function (18) does not depend on x, as the latter would just introduce a constant term without changing the set of the optimal follower solutions.

Let m denote the number of rows of matrix A, let A_j denote its j-th column, and let A_{ij} denote its generic entry. In what follows we will use notation

$$J_F := \{j \in N_x : A_j \neq 0\} \tag{22}$$

to denote the index set of the leader variables x_j (not necessarily integer-constrained) appearing in the follower problem.

Dropping the nonconvex condition (17) from (10)-(17) leads to HPR, which is a MILP in this setting. Dropping integrality conditions as well leads to the LP relaxation of HPR, which will be denoted by $\overline{\text{HPR}}$.

Solving MIBLP's is much more challenging than single-level MILP's: first of all, MIBLP's are Σ_2^P-hard [13]. Furthermore, it is known that allowing continuous variables in the leader and integer variables in the follower, may lead to bilevel problems whose optimal solutions are unattainable (see, e.g., [12, 16]). Finally, in contrast to single-level MILP's, unboundness of a relaxation of the problem (namely, the $\overline{\text{HPR}}$-relaxation) does not allow to draw conclusions on the optimal solution of MIBLP. More precisely, MIBLP's with unbounded $\overline{\text{HPR}}$-relaxation value can be unbounded, infeasible, or admit an optimal solution.

To deal with the above difficulties, in the remainder of this article we impose the following assumptions:

Assumption 1. *All the integer-constrained variables x and y have finite lower and upper bounds both in HPR and in the follower MILP.*

Assumption 2. *$J_F \subseteq J_x$, i.e., continuous leader variables x_j (if any) do not appear in the follower problem—hence they are immaterial for the computation of the value function $\Phi(x)$.*

3 A finitely-convergent branch-and-bound algorithm

We next show how a classical Branch-and-Bound (B&B) scheme for MILP can be modified to obtain an exact MIBLP solver. Correctness and finite convergence will be proved under the Assumptions 1 and 2 stated above.

If for all HPR solutions, the follower MILP is unbounded, one may safely conclude that the MIBLP is infeasible. In Section 3.1 we first show that such a situation may be detected by solving a single LP. This check can be seen as a preprocessing step, so that without loss of generality we may assume that for an arbitrary HPR solution, the follower MILP is well defined.

In Sections 3.2 and 3.3 we detail the bilevel-feasibility check and refinement procedures, that need to be "plugged-in" at some critical branch-and-bound nodes. Observe that under our assumptions, the HPR feasible set can be unbounded (we impose no bounds on continuous variables), which may result into unbounded $\overline{\text{HPR}}$ solution value. We will show that our algorithm is able to detect whether a given MIBLP is in fact unbounded, infeasible, or admits an optimal solution.

3.1 Dealing with infeasible/unbounded follower MILP's

We now address the special cases where, for a given x, function $\Phi(x)$ is not well-defined as the follower MILP (18)-(21) is either infeasible or unbounded.

In case of infeasibility, no bilevel-feasible solution of the type $(x, \cdot)$ exists. This situation of course cannot occur when (x, y) is a feasible HPR solution, in that HPR includes all follower constraints (19)–(21).

The situation where the follower MILP is unbounded for some HPR solution (x, y) was instead analyzed by [19, Lemma 2], who showed that this implies infeasibility of the MIBLP model (10)-(17). We next prove a more general result along the same lines, showing that the solution of a single LP (as opposed to a series of MILP's) is enough to handle all cases of unboundedness of the follower MILP.

Theorem 1. *Assume B is a rational matrix, and let v^* be the optimal value of the following LP:*

$$v^* := \min d^T \Delta y \tag{23}$$
$$B\Delta y \leq 0 \tag{24}$$
$$\Delta y_j \leq 0, \qquad \forall j \in \{1, \dots, n_2\} : u_j < +\infty \tag{25}$$
$$\Delta y_j \geq 0, \qquad \forall j \in \{1, \dots, n_2\} : l_j > -\infty \tag{26}$$
$$-1 \leq \Delta y \leq 1, \tag{27}$$

If $v^* < 0$, then the MIBLP model (10)-(17) is infeasible, otherwise the value function $\Phi(x)$ is well defined for every HPR solution (x, y).

Proof. Observe that the LP (23)–(27) is not unbounded (because of (27)) nor infeasible (as $\Delta y = 0$ is a feasible solution). So let Δy^* be any optimal extreme solution of it. The rationality of B implies that of Δy^*, so we can multiply the latter by a positive scalar to get an *integer* vector $\overline{\Delta y}$ that satisfies (24)–(26). Now take any HPR solution (x, y).

If $v^* < 0$, we have $d^T \overline{\Delta y} < 0$. Thus, for any solution y' satisfying (19)–(21), the new solution $y'' = y' + \alpha \overline{\Delta y}$ is feasible, and has a smaller value, for every *integer* $\alpha > 0$. This means that the follower MILP is unbounded.

If $v^* \geq 0$, instead, the LP relaxation the follower MILP cannot be unbounded, as any unbounded ray would correspond to a solution Δy of (23)–(27) with $d^T \Delta y < 0$. So the follower MILP itself cannot be unbounded, and $\Phi(x)$ is well defined.

The claim then follows from the arbitrariness of (x, y).

□

Note that the case $v^* \geq 0$ does not rule out the possibility that MIBLP model (10)-(17) is infeasible. However, Theorem 1 can be used as a preprocessing step to exclude unboundedness of all follower MILP's. In what follows, we will therefore assume that the value function $\Phi(x)$ is well defined for every HPR solution (x, y).

3.2 Feasibility check and refinement procedure

We now describe two procedures that are instrumental for the correctness of our B&B scheme.

Given a feasible solution (x^*, y^*) of HPR, checking bilevel feasibility requires solving the follower MILP (18)-(21) for $x = x^*$ to compute $\Phi(x^*)$, and to check whether $d^T y^* \leq \Phi(x^*)$ holds.

Observe that the follower MILP for $x = x^*$ cannot be infeasible, as the input $(x^*, .)$ is a feasible HPR solution, nor unbounded by assumption (see Subsection 3.1)

In a more general setting, one is interested in the following problem: given an HPR solution $(x^*, \cdot)$, find a bilevel-feasible HPR solution of the form $(\hat{x}, \cdot)$ with "$\hat{x}$ close to x^*" that minimizes the HPR objective (10), if any. This problem can be solved by the *refinement procedure* described below.

Algorithm 1: Refinement procedure

Input : An HPR solution (x^*, y^*);
Output: A refined bilevel-feasible solution $(\hat{x}, \hat{y})$ with $\hat{x}_j = x^*_j$ for all $j \in J_F$ (if any);

1 Solve the follower MILP (18)-(21) for $x = x^*$ to compute $\Phi(x^*)$;
2 Define a restricted HPR by temporarily adding the following constraints to HPR: $x_j = x^*_j$ for all $j \in J_F$, and $d^T y \leq \Phi(x^*)$;
3 Solve the restricted HPR;
4 **if** *the restricted HPR is unbounded* **then return** "MIBLP is UNBOUNDED";
5 Let $(\hat{x}, \hat{y})$ be the optimal solution found, and **return** $(\hat{x}, \hat{y})$

At Step 1 we solve the follower problem for $x_j = x^*_j$ for all $j \in J_F$. At Steps 2 and 3, one defines a restricted HPR and computes a best bilevel-feasible solution $(\hat{x}, \hat{y})$ with $\hat{x}_j = x^*_j$ for all $j \in J_F$, respectively; this is just a MILP because, for the restricted HPR, (17) becomes $d^T y \leq \Phi(x^*) = \text{const}$.

Special cases arise when the restricted HPR defined at Step 2 is either unbounded or infeasible. In the former case (Step 4), the bilevel problem (10)-(17) is unbounded as we can exhibit a bilevel-feasible solution $(\hat{x}, \hat{y})$ of arbitrarily small cost $c_x^T \hat{x} + c_y^T \hat{y}$. In the latter case, no bilevel-feasible HPR solution of the form $(\hat{x}, \cdot)$ exists, and the procedure returns a dummy solution of HPR-cost conventionally set to $+\infty$.

3.3 The overall branch-and-bound framework

Recall that our main goal is to solve MIBLP by using a standard LP-based B&B algorithm applied to HPR, where the value-function constraint (17) is handled implicitly. In our basic scheme, branching is performed

by restricting the domain of the integer-valued x and y variables, i.e., by modifying the lower/upper HPR bounds $(x_j^-, x_j^+, y_j^-, y_j^+,)$ appearing in (13) and (14) for the integer-constrained variables x_j with $j \in J_x$ and y_j with $j \in J_y^L$.

Finite convergence of the above scheme can be guaranteed in case:

a) The B&B algorithm for the underlying HPR, as well as for the follower MILP, is finely convergent.

b) One can always prune a given B&B node where all variables x_j with $j \in J_x$ have been fixed by branching.

Points a) and b) are in fact handled by Assumptions 1 and 2. A simple B&B algorithm that solves MIBLP in a finite number of iterations is then illustrated in Algorithm 2. For the sake of conciseness, only the bilevel-specific features are described in full details.

Algorithm 2: A basic branch-and-bound scheme for MIBLP

Input : A MIBLP instance satisfying Assumptions 1 and 2;
Output: An optimal MIBLP solution (if any).

1 Apply a standard LP-based B&B to HPR, branching as customary on integer-constrained variables x_j and y_j that are fractional at the optimal LP solution; incumbent update, as well as node-fathoming because of unboundedness of $\overline{\text{HPR}}$, are instead inhibited as they require a bilevel-specific check;
2 **for** *each unfathomed B&B node where standard branching cannot be performed* **do**
3 **if** *$\overline{HPR}$ is not unbounded* **then**
4 Let (x^*, y^*) be the HPR solution at the current node;
5 Compute $\Phi(x^*)$ by solving the follower MILP for $x = x^*$;
6 **if** $d^T y^* \leq \Phi(x^*)$ **then**
7 The current solution (x^*, y^*) is bilevel feasible: update the incumbent, fathom the current node, and **continue** with another node
8 **end**
9 **end**
10 **if** *all variables x_j with $j \in J_F$ are fixed by branching* **then**
11 Apply refinement Algorithm 1 to $(x, \cdot)$;
12 Possibly update the incumbent with the resulting solution $(\hat{x}, \hat{y})$, if any;
13 Fathom the current node
14 **else**
15 Branch on any x_j $(j \in J_F)$ not fixed by branching yet (even if x_j^* is integer in the LP-solution at the node), so as to reduce its domain in both child nodes
16 **end**
17 **end**

In Steps 2-17, we handle B&B nodes in which the standard branching on a fractional variable is not possible. This may happen because integrality requirements are met, or the current $\overline{\text{HPR}}$ is unbounded. Nodes for which $\overline{\text{HPR}}$ is infeasible do not need any special treatment instead, and can be fathomed as usual.

As customary, the current node is fathomed in case the current relaxation is finite and has an optimal solution that is bilevel feasible (Step 7).

Compared to a standard B&B algorithm, the following modified rules need to be applied:

- Fathoming is inhibited in case of an unbounded $\overline{\text{HPR}}$ value. Instead, at Step 15 one continues to branch on an integer x_j (even if x_j^* is integer) until its value is fixed by branching.

- If $\overline{\text{HPR}}$ value is unbounded and all variables x_j $(j \in J_F)$ are fixed by branching, we apply the refinement algorithm of Step 11. If the refinement procedure returns "MIBLP is UNBOUNDED", the algorithm

terminates. Otherwise, the incumbent is possibly updated (if the refinement procedure returns a finite value), and the node is eventually fathomed.

- Branching is also not required if all variables x_j ($j \in J_F$) have been fixed by branching (Step 10). Indeed, under Assumption 2, one can use Algorithm 1 to compute the best bilevel-feasible solution $(\hat{x}, \hat{y})$ for the node.

As customary, the algorithm returns "MIBLP is INFEASIBLE", if upon the enumeration of all B&B nodes, no feasible solution is found. Observe that, even in case of an unbounded $\overline{\text{HPR}}$ value, this situation is handled correctly, due to the refinement algorithm of Step 11. In the worst case, this step will be executed for every possible combination of feasible x_j integer values, $j \in J_F$, until all of the B&B nodes are discarded.

Theorem 2. *Under Assumptions 1 and 2, Algorithm 2 correctly solves MIBLP in a finite number of iterations.*

Proof. Finiteness follows immediately Assumption 1, as each branching operation strictly reduces the domain of an integer-constrained variable. Thus a finite number of B&B nodes will be generated, and each node requires a finite number of operations to be processed. Correctness follows from the fact that, because of Assumption 2, Step 11 actually computes the best bilevel-feasible solution $(\hat{x}, \hat{y})$ for the current node (if any), so the node can be pruned after the incumbent update.

□

Note that we allow branching on y variables, though this could be avoided by a simple modification of our scheme. The rationale of this choice is that we prefer to exploit the underlying MILP solver as much as possible, and to deviate from it only when strictly needed for the correctness of the approach. Also observe that, contrarily to other approaches from the literature, branching on y variables is a legitimate option in our setting because it does not affect the follower lower/upper bound vectors l and u, hence value function $\Phi(x^*)$ computed at Step 4 remains "completely blind" with respect to branching.

3.4 Comparison with the literature

The first generic branch-and-bound approach to MIBLP was given in [16]. The approach works for problems without y variables in the leader constraints. Our algorithm is much less restrictive than the one in [16], for which the authors are able to prove convergence only under the assumptions that the HPR feasible region is compact and that either all leader variables are integer (i.e., $J_x = N_x$), or all follower variables are continuous (i.e., $J_y^L = J_y = \emptyset$).

A MILP-based branch-and-cut algorithm was introduced in [6, 7]. This approach builds upon the ideas from [16], and cuts off integer bilevel infeasible solutions by adding cuts that exploit the integrality property of the leader and the follower variables. However, this method works with integer variables only and does not allow y variables in the leader constraints.

In the branch-and-sandwich in [11] (which is a more general approach for nonlinear bilevel problems) novel ideas for deriving lower and upper bounds on the follower value function $\Phi(x)$ are proposed. Also here, it is assumed that the HPR feasible region is compact. On the other hand, continuous x variables influencing the follower's decision are allowed (i.e., $J_F \subseteq N_x$), in which case only ϵ-convergence can be guaranteed.

In [19], an exact approach based on multi-way branching on the slack variables on the follower constraints is proposed. The algorithm solves a series of MILP's, obtained by restricting the slack variables, until the convergence is reached. Again, all x variables are required to be integer and bounded, and the constraint matrix A must be integer.

Recently, [5] proposed a method that works for integer x and y variables only: HPR is embedded into a branch-and-bound tree, bilevel infeasible solutions being cut off by linearizing the bilevel continuous problems resulting from the separation procedure.

A Benders-like decomposition scheme for general MIBLP's is given in [18], whereas an approach for nonlinear MIBLP's is introduced in [15]; in both cases, the algorithms assume the HPR feasible region be compact.

To the best of our knowledge, our algorithm is a first convergent branch-and-bound approach which returns a provably optimal solution (if such exists) without assuming the HPR feasible region to be compact (only integer variables need to be bounded). Moreover, our algorithm is capable of handling continuous x variables (as long as they do not influence the follower decisions), in combination with a mixed-integer follower.

4 A branch-and-cut algorithm

We next elaborate the B&B algorithm described in the previous section, and introduce new families of linear cuts used to speedup its convergence within a sound Branch&Cut (B&C) scheme.

4.1 Intersection cuts

Intersection Cuts (IC's) have been introduced by Balas [3] in the 1970th, and are widely used in Mixed-Integer Programming. Their use for MIBLP was not investigated until our very recent work [10], where for the first time we showed their usefulness in the context of bilevel optimization.

The definition of an IC violated by a given point (x^*, y^*) requires the definition of two sets:

(1) a cone pointed at (x^*, y^*) that contains all the bilevel feasible solutions, and

(2) a convex set S that contains (x^*, y^*) but no bilevel feasible solutions in its interior.

The larger the convex set, and the smaller the cone, the better the resulting IC. We will next briefly address point (1), while point (2) will be the subject of the next section.

As customary in mixed-integer programming, our IC's are generated for vertices (x^*, y^*) of the $\overline{\text{HPR}}$ polyhedron, so a suitable cone is just the corner polyhedron associated with the corresponding optimal basis. All relevant information about this cone is readily available in the "optimal tableau". As the $\overline{\text{HPR}}$ at a given B&B node exploits locally-valid information (notably, the reduced variable domain resulting from branching), our IC's will be locally (as opposed to globally) valid as well.

4.2 Bilevel-free polyhedra

We next describe how to derive bilevel-free polyhedra to be used to generate valid cuts as outlined in the previous section. With a little abuse of notation, in what follows we will call "facet" an inequality appearing in the outer description of a polyhedron.

Theorem 3 below was implicit in some early references (including [19]), where it was only used as a branching rule in a B&B setting.

Theorem 3. *For any $\hat{y} \in \mathbb{R}^{n_2}$ that satisfies (20)–(21), the set*

$$S(\hat{y}) = \{(x, y) \in \mathbb{R}^n : d^T y > d^T \hat{y},\ Ax + B\hat{y} \leq b\} \tag{28}$$

does not contain any bilevel feasible point (not even on its frontier).

Proof. We have to prove that no bilevel feasible (x, y) exists such that $d^T y > d^T \hat{y}$ and $Ax + B\hat{y} \leq b$. Indeed, for any bilevel feasible solution (x, y) with $Ax + B\hat{y} \leq b$, one has

$$d^T y \leq \Phi(x) = \min_{y'} \{d^T y' : Ax + By' \leq b,\ (20)\text{–}(21) \text{ hold for } y'\} \leq d^T \hat{y},$$

hence $(x, y) \notin S(\hat{y})$.

□

Note that inequalities $(Ax+By)_i \le b_i$ in the follower MILP that do not involve x variables, if any, would lead to useless inequalities of the form $0 \le b_i - (B\hat{y})_i$ (= a nonnegative constant), so they do not contribute to the definition of $S(\hat{y})$. This is the reason why, for example, the bound constraints $l \le y \le u$ do not correspond to facets of $S(\hat{y})$.

Unfortunately, the above set is not directly suited to be used for IC generation, as one needs to ensure that any bilevel-infeasible HPR solution (x^*, y^*) to be cut belongs to the *interior* of the bilevel-free polyhedron—actually, to get numerically reliable IC's one has to guarantee a certain minimum distance of (x^*, y^*) from all the facets of the polyhedron itself. So one needs to address *extended* versions of the above set, whose facets are "moved apart" by a non-negligible amount. In our setting, this can be done under the following assumption.

Assumption 3. *$Ax + By - b$ is integer for all HPR solutions (x, y).*

The above assumption holds true, in particular, when all the x and y variables appearing in the follower MILP are constrained to be integer, and (A, B, b) is integer—a very common assumption in the MIBLP literature.

Let $\mathbf{1} = (1, \cdots, 1)$ denote a vector of all ones of suitable size.

Theorem 4. *Under Assumption 3, for any $\hat{y} \in \mathbb{R}^{n_2}$ that satisfies (20)–(21), the extended polyhedron*

$$S^+(\hat{y}) = \{(x, y) \in \mathbb{R}^n : d^T y \ge d^T \hat{y},\ Ax + B\hat{y} \le b + \mathbf{1}\} \tag{29}$$

does not contain any bilevel feasible point in its interior.

Proof. To be in the interior of $S^+(\hat{y})$, a bilevel feasible (x, y) should satisfy $d^T y > d^T \hat{y}$ and $Ax + B\hat{y} < b + \mathbf{1}$. By assumption, the latter condition can be replaced by $Ax + B\hat{y} \le b$, hence the claim follows from Theorem 3. □

Example. Figure 1 illustrates the application of our IC's to a notorious example from [16] which is frequently used in the literature, namely:

$$\min_{x \in \mathbb{Z}} \ -x - 10y \tag{30}$$

$$y \in \arg\min_{y' \in \mathbb{Z}} \{\ y' : \tag{31}$$

$$-25x + 20y' \le 30 \tag{32}$$

$$x + 2y' \le 10 \tag{33}$$

$$2x - y' \le 15 \tag{34}$$

$$2x + 10y' \ge 15\ \}. \tag{35}$$

In this all-integer example, there are 8 bilevel feasible points (depicted as crossed squares in Figure 1), and the optimal bilevel solution is $(2, 2)$. The drawn polytope corresponds to the $\overline{\text{HPR}}$ feasible set.

We first apply the definition of the bilevel-free set from Theorem 3 with $\hat{y}$ defined as the follower optimal solution for $x = x^*$. After solving the first $\overline{\text{HPR}}$, the point $A = (2, 4)$ is found. This point is bilevel infeasible, as for $x^* = 2$ we have $d^T y^* = y^* = 4$ while $\Phi(x^*) = 2$. Solving the follower for $x = 2$ we compute $\hat{y} = 2$ and the intersection cut derived from the associated $S(\hat{y})$ is depicted in Figure 1(a). In the next iteration, the optimal $\overline{\text{HPR}}$ solution moves to $B = (6, 2)$. Again, for $x^* = 6$, $f(x^*, y^*) = y^* = 2$ while $\Phi(x^*) = 1$. So we compute $\hat{y} = 1$ and generate the IC induced by the associated $S(\hat{y})$, namely $2x + 11y \le 27$ (cf. Figure 1(b)). In the next iteration, the fractional point $C = (5/2, 2)$ is found and $\hat{y} = 1$ is again computed. In this case, C is not in the interior of $S(\hat{y})$ so we cannot generate an IC cut from C but we should proceed and optimize HPR to integrality by using standard MILP tools such as MILP cuts or branching. This produces the optimal HPR solution $(2, 2)$ which is bilevel feasible and hence optimal.

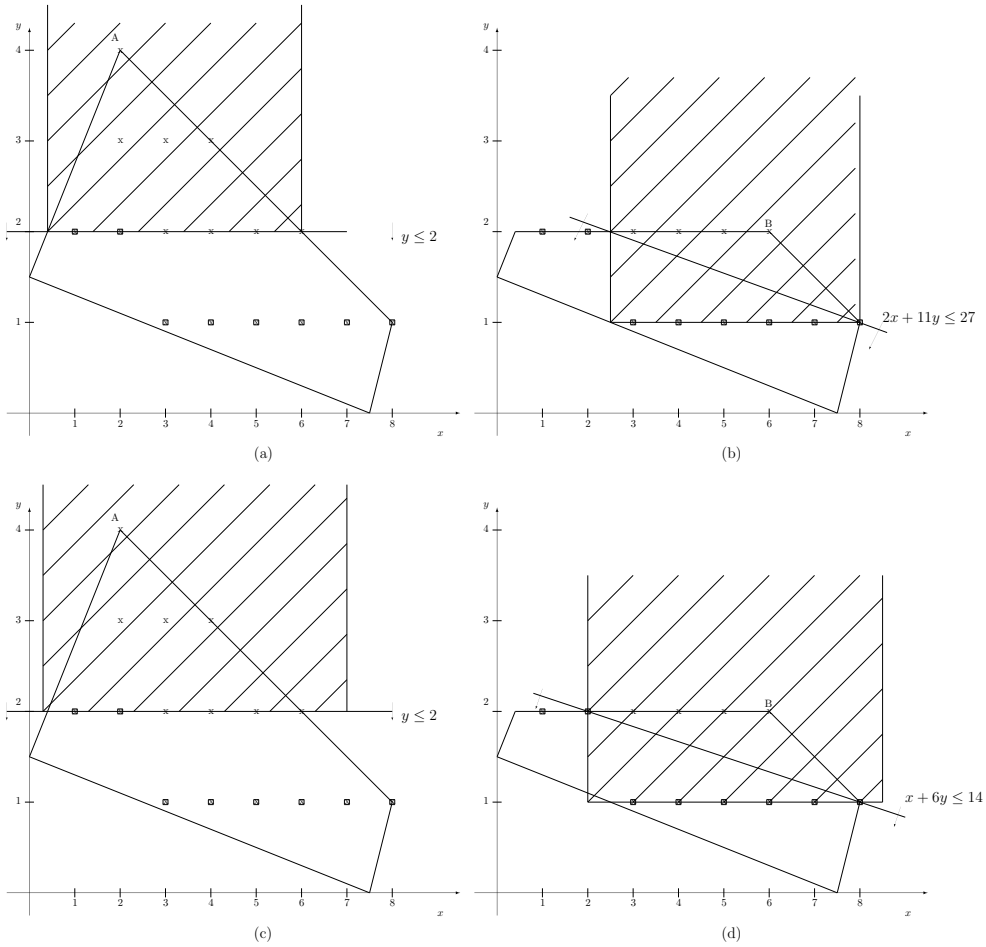


Figure 1: Illustration of the effect of alternative intersection cuts for a notorious example from [16]. Shaded regions correspond to the bilevel-free sets for which the cut is derived.

We next apply the definition of the enlarged bilevel-free set from Theorem 4 (whose assumption is fulfilled in this example) with $\hat{y}$ defined as before; see Figures 1(c) and (d). After the first iteration, the point $A = (2, 4)$ is cut off by a slightly larger $S^+(\hat{y} = 2)$, but with the same IC as before ($y \le 2$). After the second iteration, from the bilevel infeasible point $B = (6, 2)$ we derive a larger set $S^+(\hat{y} = 1)$ and a stronger IC ($x + 6y \le 14$). In the third iteration, solution $D = (2, 2)$ is found which is the optimal bilevel solution, so no branching at all is required in this example.

4.3 Relaxed bilevel-free polyhedra

We now address a very crucial point for the computational effectiveness of IC's, namely: the bilevel-free polyhedron should be as large as possible in order to derive sufficiently deep IC's. In particular, one should try to "remove from the polyhedron" as many facets as possible—as long as this does not affect the bilevel-free property of course. To this end, one can apply the following simple (yet very powerful) argument.

Theorem 5. *Let $S = \{(x, y) \in \mathbb{R}^n : \alpha_i^T x + \beta_i^T y \le \gamma_i, i = 1, \dots, k\}$ be any polyhedron not containing bilevel-feasible points in its interior. Then one can remove from S all its facets $i \in \{1, \dots, k\}$ such that the half-space $\{(x, y) \in \mathbb{R}^n : \alpha_i^T x + \beta_i^T y \ge \gamma_i\}$ does not contain any bilevel-feasible solution.*

Proof. Obvious, as the removal of any such facet from the definition of S cannot bring bilevel-feasible points into it.

□

Note that the condition on the theorem refers to a closed half-space, i.e., the condition is $\geq \gamma_i$ and not $> \gamma_i$. This is consistent with the fact that S is allowed to contain bilevel-feasible points on its frontier.

Corollary 1. *Let $S = \{(x, y) \in \mathbb{R}^n : \alpha_i^T x + \beta_i^T y \leq \gamma_i,\ i = 1, \ldots, k\}$ be any polyhedron not containing bilevel-feasible points in its interior. Then one can remove from S all its facets $i \in \{1, \ldots, k\}$ such that*

$$\sum_{j=1}^{n_1} \max\{\alpha_{ij} x_j^-, \alpha_{ij} x_j^+\} + \sum_{j=1}^{n_2} \max\{\beta_{ij} y_j^-, \beta_{ij} y_j^+\} < \gamma_i.$$

In our implementation, the above corollary is automatically applied within IC separation, by using the current lower/upper bounds (x^-, x^+, y^-, y^+) at the given B&B node.

Corollary 2. *Let $S^+(\hat{y})$ be the bilevel-free polyhedron defined by (29) and assume a lower bound FLB on $d^T y$ is known. If $d^T \hat{y} < FLB$, one can remove the facet $d^T y \geq d^T \hat{y}$ from the definition of $S^+(\hat{y})$.*

In our implementation, the above property is only exploited for "zero-sum" instances where the leader and follower objective functions satisfy $F(x, y) = -f(x, y)$, i.e., $c_x = 0$ and $c_y = -d$ in (10). Indeed, at every B&B node, the incumbent value z^* (say) is an upper bound for the leader objective function, hence $FLB = -z^*$ is a lower bound for the follower's one.

4.4 Separation of intersection cuts

We now address the question of how to cut a given vertex (x^*, y^*) of $\overline{\text{HPR}}$ by using an IC. Given for granted that we use the cone associated with the current LP basis, as explained in Subsection 4.1, what remains is the choice of the bilevel-free set to be used.

As IC's require that (x^*, y^*) belongs to the interior of the bilevel-free set, we concentrate on the extended polyhedron $S^+(\hat{y})$ from Theorem 4. In doing so, we have to restrict ourselves to the cases where Assumption 3 holds. We also assume that (possibly after scaling) d is an integer vector, so as to replace the bilevel-infeasibility conditions of the type $d^T y < d^T y^*$ by $d^T y \leq d^T y^* - 1$.

As our IC's are locally valid cuts, in this section we denote by (x^-, x^+, y^-, y^+) the variable bounds at the current branching node, as modified by branching. Needless to say, notation (A, B, b, d, l, u) refers instead to the original follower MILP, that is required to be completely blind with respect to branching.

A very natural option for defining $S^+(\hat{y})$ is to choose $\hat{y}$ as an optimal solution of the follower MILP for $x = x^*$, namely:

$$\mathbf{SEP-1}: \quad \hat{y} \in \arg\min d^T y \tag{36}$$

$$Ax^* + By \leq b \tag{37}$$

$$l \leq y \leq u \tag{38}$$

$$y_j \text{ integer} \qquad \forall j \in J_y \tag{39}$$

Note that the above problem is always feasible when (x^*, y^*) is an HPR solution. Also note that one cannot write $\leq b + \mathbf{1}$ in (37) as this would allow $(x^*, \cdot)$ to belong to the frontier of $S^+(\hat{y})$.

In our B&C scheme, the computation of $\hat{y}$ typically does not require extra effort as the follower MILP is solved anyway for the sake of checking bilevel feasibility of (x^*, y^*) and, possibly, producing heuristic solutions.

By minimizing $d^T \hat{y}$, the above model maximizes the distance of (x^*, y^*) from the facet $d^T y \geq d^T \hat{y}$ of $S^+(\hat{y})$. This is an aggressive policy that works very well in some cases. However, a possible drawback is that the other facets of $S^+(\hat{y})$ can be quite close to (x^*, y^*), which tends to reduce the depth of the derived IC.

An alternative approach is to define $\hat{y}$ so as to have a large number of "removable facets" according to Corollary 1, i.e., facets of type $(Ax + B\hat{y})_i \leq b_i + 1$ with

$$\sum_{j \in N_x} \max\{A_{ij}x_j^-, A_{ij}x_j^+\} + (B\hat{y})_i \leq b_i. \tag{40}$$

This leads to the following alternative separation MILP (recall that m denotes the number of rows of B).

$$\begin{aligned}
\mathbf{SEP-2}: \quad \hat{y} \in \arg\min & \sum_{i=1}^{m} w_i && && (41)\\
& d^T y \leq d^T y^* - 1 && && (42)\\
& By + s = b && && (43)\\
& s_i + (L_i^{max} - L_i^*)\, w_i \geq L_i^{max}, && \forall i = 1, \ldots, m && (44)\\
& l \leq y \leq u && && (45)\\
& y_j \text{ integer}, && \forall j \in J_y && (46)\\
& s \text{ free} && && (47)\\
& w \in \{0,1\}^m && && (48)
\end{aligned}$$

where, for each $i = 1, \ldots, m$,

$$L_i^* := \sum_{j \in N_x} A_{ij}x_j^* \;\leq\; L_i^{max} := \sum_{j \in N_x} \max\{A_{ij}x_j^-, A_{ij}x_j^+\}.$$

In the model, the binary variable w_i attains value 0 if the facet $(Ax+B\hat{y})_i \leq b_i+1$ can be removed according to (40), $w_i = 1$ otherwise.

The objective function (41) then minimizes the number of facets that cannot be removed.

Equations (43) define the free variable $s_i = (b - By)_i$ for each constraint, hence condition $s \geq L^*$ implied by (44) actually imposes that the final solution $\hat{y}$ satisfies $Ax^* + B\hat{y} \leq b$. Together with (42), this guarantees that (x^*, y^*) belongs to the interior of $S^+(\hat{y})$.

In case $w_i = 0$, constraint (44) actually enforces the stronger condition $s_i \geq L_i^{max}$, i.e., $\sum_{j \in N_x} \max\{A_{ij}x_j^-, A_{ij}x_j^+\} + (B\hat{y})_i \leq b_i$. Hence condition (40) holds and the corresponding facet can be removed, as claimed.

4.5 Informed no-good cuts

A known drawback of IC's is their dependency on the LP basis associated with the point to cut, which can create cut accumulation in the LP relaxation and hence shallow cuts and numerical issues. Moreover, IC's are not directly applicable if the point to cut is not a vertex of a certain LP relaxation of the problem at hand, as it happens e.g. when it is computed by the internal MILP heuristics.

We next describe a general-purpose variant of IC's whose derivation does not require any LP basis and is based on the well-known interpretation of IC's as disjunctive cuts. In a sense, we are replacing the cone defined by the LP basis by the cone induced by the tight bound constraints. It turns out that the resulting inequality is valid and violated by any bilevel infeasible solution of HPR in the relevant special case where all x and y variables are binary.

We are given a point $\xi^* = (x^*, y^*) \in \mathbb{R}^n$ and a polyhedron

$$S = \{\xi \in \mathbb{R}^n : g_i^T \xi \leq g_{i0},\, i = 1, \ldots, k\}$$

whose interior contains ξ^* but no bilevel-feasible points. Assume that variable-bound constraints $\xi^- \leq \xi \leq \xi^+$ are present in the $\overline{\text{HPR}}$, where some entries of ξ^- or ξ^+ can be $-\infty$ or $+\infty$, respectively. Given ξ^* with $\xi^- \leq \xi^* \leq \xi^+$, let

$$L := \{j \in \{1, \ldots, n\} : |\xi_j^* - \xi_j^-| \leq |\xi_j^* - \xi_j^+|\}$$

and $U := \{1,\dots,n\} \setminus L$. In other words, L contains the indices of the variables that are closer to their lower bound than to their upper bound, and vice-versa for U. For the relevant case where $\xi^*_j \in \{\xi^-_j, \xi^+_j\}$ for all $j \in \{1,\dots,n\}$, sets L and U then contain the indices of the variables at their lower or upper bound, respectively. Given the partition (L,U), we define the corresponding linear mapping $\xi \mapsto \overline{\xi} \in \mathbb{R}^n$ with

$$\overline{\xi}_j := \begin{cases} \xi_j - \xi^-_j, & \text{for } j \in L \\ \xi^+_j - \xi_j, & \text{for } j \in U \end{cases}$$

(variable shift and complement). By assumption, any feasible point ξ must satisfy the k-way disjunction

$$\bigvee_{i=1}^{k} \Big(\sum_{j=1}^{n} g_{ij}\xi_j \geq g_{i0}\Big), \tag{49}$$

whereas ξ^* violates all the above inequalities. Now, each term of (49) can be rewritten in terms of $\overline{\xi}$ as

$$\sum_{j=1}^{n} \overline{g}_{ij}\,\overline{\xi}_j \geq \overline{\beta}_i := g_{i0} - \sum_{j\in L} g_{ij}\xi^-_j - \sum_{j\in U} g_{ij}\xi^+_j, \tag{50}$$

with $\overline{g}_{ij} := g_{ij}$ if $j \in L$, $\overline{g}_{ij} = -g_{ij}$ otherwise. If $\overline{\beta}_i > 0$ for all $i = 1,\dots,k$, one can normalize the above inequalities to get $\sum_{j=1}^{n}(\overline{g}_{ij}/\overline{\beta}_i)\,\overline{\xi}_j \geq 1$ and derive the valid disjunctive cut in the $\overline{\xi}$ space

$$\sum_{j=1}^{n} \overline{\gamma}_j \overline{\xi}_j \geq 1, \tag{51}$$

where $\overline{\gamma}_j := \max\{\overline{g}_{ij}/\overline{\beta}_i : i = 1,\dots,k\}$. Finally, one can transform it back to the ξ space in the obvious way.

It is easy to see that, in case $\xi^*_j \in \{\xi^-_j, \xi^+_j\}$ for all $j \in \{1,\dots,n\}$, one has $\overline{\beta} > 0$, hence the above cut is indeed valid and obviously violated as $\overline{\xi}^* = 0$. In all other cases, the above cut separation is just heuristic.

Inequality (51) will be called *Informed No-Good* (ING) cut as it can be viewed as a strengthening of the following *no-good* cut often used for bilevel problems with all-binary variables—and in many other Constraint Programming (CP) and Mathematical Programming (MP) contexts:

$$\sum_{j\in L} \xi_j + \sum_{j\in U} (1 - \xi_j) \geq 1. \tag{52}$$

The cut above corresponds to the very generic choice

$$S = \{\xi \in \mathbb{R}^n : \xi_j \leq 1\,\forall j \in L,\ 1 - \xi_j \leq 1\,\forall j \in U\}$$

and is violated by ξ^* but is satisfied by any other binary point, hence resulting into a very weak cut. To the best our knowledge, ING cuts are new; they will hopefully be useful in other CP and MP contexts.

5 Computational results

To evaluate the performance of our B&C solution method, we implemented it (in C language) on top of the general-purpose MILP solver IBM ILOG Cplex 12.6.3 using callbacks.

Internal Cplex's heuristics as well as preprocessing have been deactivated in all experiments. IC separation is applied both to fractional solutions (in the so-called *usercut callback*) and to integer solutions (in the so-called *lazyconstraint callback*). For fractional solutions, IC's whose normalized violation is very small are just skipped, and a maximum number of cuts `max_node_cuts` (a given parameter) is allowed to be generated at each node.

Table 1: Our testbed. Column *#inst* reports the total number of instances in the class, while column *type* indicates whether the instances are binary (B) or integer (I).

Class	source	# inst	type	Notes
`CARAMIA-MARI`	[5]	70	I	randomly generated (very small and easy)
`DENEGRE`	[6],[17]	50	I	randomly generated (medium difficulty)
`INTERDICTION`	[6],[17]	125	B	interdiction inst.s (treated as general bilevel)
`MIPLIB`	[10]	57	B	from MIPLIB 3.0 (very large and difficult)

Internal numerical-precision thresholds for integrality and constraint-satisfaction tests are set to a very small value (10^{-9}) so as to guarantee a very precise overall computation.

Our computational study has been performed on an Intel Xeon E3-1220V2 3.1GHz, with 16GB of RAM. This is a quad-core processor launched by Intel in 2012 and credited for 1,892 Mflop/s in the Linpack benchmark report of Dongarra [9].

Computing times reported in what follows are in wall-clock seconds and refer to 4-thread runs. The time limit for each run was set to 3,600 wall-clock seconds.

5.1 Testbed

Table 1 summarizes details about the data sets that have been considered in our computational study. The following four data sources with a total number of 302 instances have been considered: `CARAMIA-MARI` (available from the authors of [5] upon request), `DENEGRE` and `INTERDICTION` (both available at `https://github.com/tkralphs/MibS/tree/library/data` in February 2016), and `MIPLIB` (available at `http://msinnl.github.io/pages/bilevel.html`).

Class `CARAMIA-MARI` contains 70 randomly generated instances with $n_1, n_2 \in \{5, 10, 15\}$, with no leader constraints and with $m = n_1 + n_2$ follower constraints. All variables are required to be integer and all coefficients are randomly generated integer values; see [5] for further details.

Instances of class `DENEGRE` have been proposed in [6]. They consist of $n_1 \in \{5, 10, 15\}$ integer leader variables, and the number of integer follower variables n_2 is set such that $n_1 + n_2 = 20$. There are $m \in \{20, 30, 40\}$ follower constraints and no constraints at the leader. All coefficients are integers in the range $[-50, 50]$.

Instances of class `INTERDICTION` have the following structure. All variables are binary and each variable in the leader is associated with a variable in the follower, and vice-versa. The leader has a single constraint that is called the *interdiction-budget* constraint. The follower problem consists of some combinatorial optimization problem defined on the y variables only, plus additional constraints that involve both x and y variables and have the form $x_i + y_i \leq 1$. In other words, the leader is allowed to interdict to the follower the use of some items, subject to the interdiction budget. The leader and follower problems share the same objective function, with opposite signs. In class `INTERDICTION`, the follower problem is an assignment problem (25 instances with 25+25 variables) or a knapsack problem (100 instances with up to 50+50 variables). Although interdiction problems can in some cases be treated with specialized ad-hoc algorithms exploiting the problem structure (as, e.g., done in [6]), in our computational study they are treaded as general bilevel problems.

Finally, class `MIPLIB` has been introduced in [10] with the aim of producing very challenging instances for MIBLP solvers. It is derived from 19 instances of MILPLIB 3.0 [4] containing only binary variables. They have been converted into bilevel problems by labeling the first $Y\%$ (rounded up) variables as y's, and the remaining ones as x's, with $Y \in \{10, 50, 90\}$, thus resulting in 57 instances in total. All constraints in the resulting model belong to the follower subproblem, while the objective function is used as the leader objective $c_x^T x + c_y^T y$. Finally, the follower objective is defined as $d^T y = -c_y^T y$. In [10], only 30 out of 57 instances from this class have been considered—those containing equality constraints were left out, for the sake of comparison with an alternative solver which could not handle equality constraints. By design, instances in class `MIPLIB` are much larger (and difficult) than those in the other classes, and involve up to about 80,000

HPR variables and up to about 5,000 follower constraints.

5.2 Benchmark solver

In order to have a benchmark code, we implemented the solution method recently proposed in [6], called `BENCHMARK` in what follows. To have an apple-to-apple comparison, we decided to embed the cuts proposed in [6, cf. Proposition 2.2] within our own Cplex-based code. These cuts can only be applied to pure integer MIBLP instances in which all variables are integer-constrained, and all constraint coefficients are integer as well. They are used to cut-off a bilevel-infeasible integer vertex (x^*, y^*) of $\overline{\text{HPR}}$ at a given node, so they have been implemented in the Cplex's lazyconstraint callback. Each such cut is obtained by adding up all tight constraints (including variable bounds) at (x^*, y^*), written in their $\geq$ form, and then adding 1 to the right-hand side. Note that the resulting cut is locally valid, and requires that all coefficients in the node-LP matrix are integer (meaning that one is not allowed to generate other cuts with non-integer coefficients); see [6] for details.

A comparison with the results on 30 instances (that were available online) from the set `DENEGRE` reported in [6, cf. Table 2.4] is given in Table 2. Computing times for the original implementation refer to an eight-core AMD Opteron Processor 6128 @2.0 Ghz with 32GB of memory, launched in March 2010. Time limit for `BENCHMARK` was set to 3,600 wall-clock seconds, while a time limit of 30,000 seconds was used in [6]. Table 2 reports the value of the best solution found (BestSol), the computing time in seconds (time), the percentage gap between the final lower and upper bounds (%GAP), and the speedup of `BENCHMARK` with respect to the original implementation (column *speedup*, not reported in case of both hit the time limit).

According to the table, our `BENCHMARK` implementation is about 10-100 times faster (and solves to proven optimality two more instances) than the original one. Moreover, for problems not solved to proven optimality, `BENCHMARK` systematically produces better lower and upper bounds. This is not surprising, as our implementation is based on the commercial software Cplex, while the original one uses the open-source software COIN-OR (BLIS).

We can therefore conclude that our benchmark code `BENCHMARK` represents in fair way a state-of-the-art exact solver for general MIBLP's .

5.3 Results

In this subsection we compare the performance of six alternative settings for our MIBLP solver, namely:

- `SEP-1a`: our B&C solver using model SEP-1 of Subsection 4.4 for IC separation, and generating at most `max_node_cuts`=20 cuts at each node (including root);
- `SEP-2a`: our B&C solver with SEP-2 and `max_node_cuts`=20 for all nodes (including root);
- `SEP-1b`: our B&C solver with SEP-1 and `max_node_cuts`=20 for the root node only (=0 for all other nodes);
- `SEP-2b`: our B&C solver with SEP-2 and `max_node_cuts`=20 for the root node only (=0 for all other nodes);
- `ING`: our B&C solver with ING cuts applied to the SEP-1 bilevel-free polyhedron; only integer points being separated, i.e., `max_node_cuts`=0 for all nodes;
- `BENCHMARK`: our benchmark code implementing cuts in [6].

The computational analysis reported in [10] shows that ING cuts significantly outperform standard no-good cuts, hence the latter were not considered in our computational tests. Note that, similarly to no-good cuts, ING cuts only work for binary problems, thus setting `ING` cannot be applied to the instances of classes `CARAMIA-MARI` and `DENEGRE`, as they contain general-integer variables.

Table 2: Performance of our BENCHMARK code.

	Original implementation [6]			Our BENCHMARK implementation			
Instance	BestSol	time [s]	%GAP	BestSol	time [s]	%GAP	speedup
miblp-20-20-50-0110-5-1	-548	60.62	0	-548	0.78	0	77.7
miblp-20-20-50-0110-5-2	-558	30000.00	25.1	-583	3600.00	5.7	–
miblp-20-20-50-0110-5-3	-477	0.14	0	-477	0.01	0	14.0
miblp-20-20-50-0110-5-4	-753	0.22	0	-753	0.02	0	11.0
miblp-20-20-50-0110-5-5	-392	0.11	0	-392	0.01	0	11.0
miblp-20-20-50-0110-5-6	-1061	1091.91	0	-1061	5.38	0	203.0
miblp-20-20-50-0110-5-7	-547	0.35	0	-547	0.02	0	17.5
miblp-20-20-50-0110-5-8	-936	0.32	0	-936	0.02	0	16.0
miblp-20-20-50-0110-5-9	-877	0.24	0	-877	0.02	0	12.0
miblp-20-20-50-0110-5-10	-340	0.85	0	-340	0.03	0	28.3
miblp-20-20-50-0110-10-1	-353	30000.00	47.0	-359	3600.00	30.7	–
miblp-20-20-50-0110-10-2	-659	15.82	0	-659	0.63	0	25.1
miblp-20-20-50-0110-10-3	-618	120.31	0	-618	0.89	0	135.2
miblp-20-20-50-0110-10-4	-597	30000.00	25.7	-604	3600.00	13.8	–
miblp-20-20-50-0110-10-5	-1003	0.06	0	-1003	0.01	0	6.0
miblp-20-20-50-0110-10-6	-672	30000.00	26.2	-707	3600.00	17.3	–
miblp-20-20-50-0110-10-7	-618	30000.00	36.8	-669	3600.00	22.3	–
miblp-20-20-50-0110-10-8	-667	997.46	0	-667	9.89	0	100.9
miblp-20-20-50-0110-10-9	-256	6849.91	0	-256	39.12	0	175.1
miblp-20-20-50-0110-10-10	-429	30000.00	23.6	-441	773.36	0	38.8
miblp-20-20-50-0110-15-1	-289	30000.00	60.6	-420	3600.00	31.4	–
miblp-20-20-50-0110-15-2	-645	30000.00	23.2	-645	3600.00	17.1	–
miblp-20-20-50-0110-15-3	-593	30000.00	20.2	-593	3499.93	0	8.6
miblp-20-20-50-0110-15-4	-396	30000.00	36.4	-424	3600.00	20.9	–
miblp-20-20-50-0110-15-5	-75	30000.00	90.1	-320	3600.00	52.2	–
miblp-20-20-50-0110-15-6	-596	30000.00	40.4	-596	3600.00	32.2	–
miblp-20-20-50-0110-15-7	-471	30000.00	28.0	-471	3600.00	3.5	–
miblp-20-20-50-0110-15-8	-242	30000.00	73.9	-301	3600.00	64.8	–
miblp-20-20-50-0110-15-9	-584	324.33	0	-584	4.30	0	75.4
miblp-20-20-50-0110-15-10	-251	9.12	0	-251	0.08	0	114.0

Table 3: Summary of results obtained for the CARAMIA-MARI class. All 70 instances are solved to optimality by each of the settings. Average computing time ($t[s]$) and average number of nodes are reported.

setting	SEP-1a	SEP-2a	SEP-1b	SEP-2b	BENCHMARK
$t[s]$	1.5	2.5	0.1	0.3	0.1
nodes	88.2	101.1	369.7	345.0	398.5

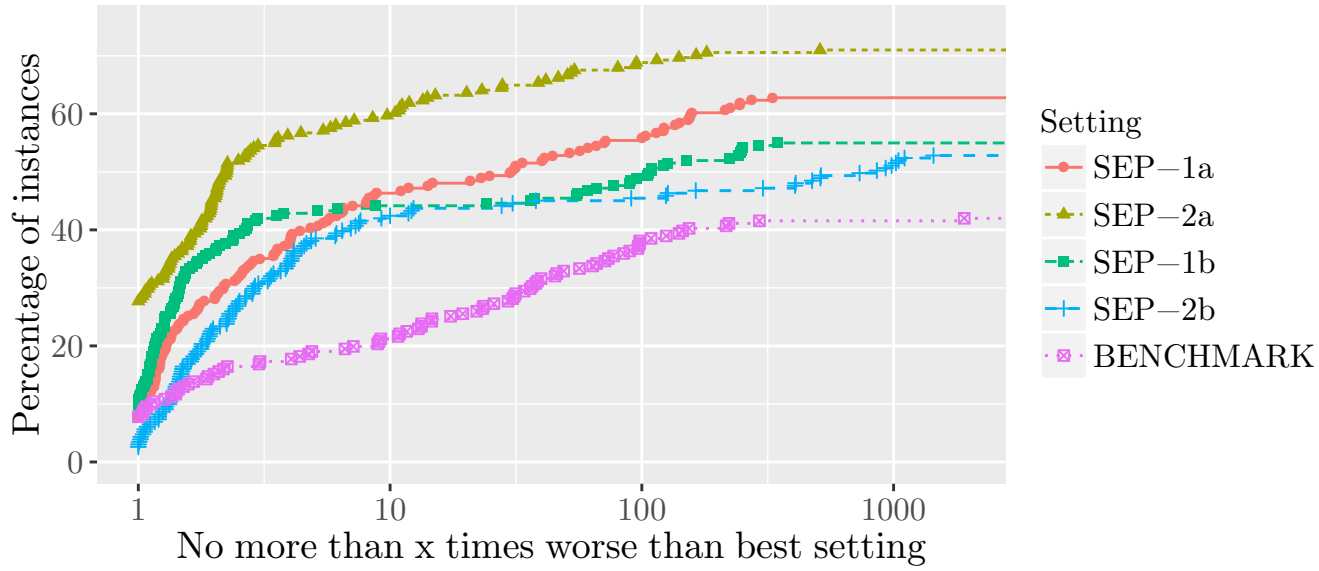
First, we provide a summary of our results on the class of CARAMIA-MARI instances, see Table 3. Given that these instances are solved by each of our settings within fractions of a second, we conclude that they are too easy and do not consider them in the remainder of this computational study. Note that the best performing approach presented in [5] needed on average 35 seconds for this class, on a PC Pentium Core 2 Duo with a 2 GHz processor and 1 GB RAM and using CPLEX 12.3.

We next compare the performance of our different settings using performance profiles [8]. To construct performance profiles, for each setting $s \in S$ and instance $p \in P$, a *performance ratio*

$$r_{p,s} = \frac{t_{p,s}}{\min_{s' \in S}\{t_{p,s'}\}}$$

is calculated, where $t_{p,s}$ is the time setting s needs to solve instance p to optimality. If a setting s does not solve instance p, $r_{p,s}$ is set to r_M, which is a value larger than any $r_{p,s}$, e.g., $r_M = \max_{s \in S, p \in P} r_{p,s} + 1$. In

Figure 2: Performance profile plot over all instances (classes DENEGRE, INTERDICTION and MIPLIB); setting ING is missing as it only works for binary problems. For each plotted line, the leftmost value gives the percentage of instances for which the corresponding setting was the fastest, while the rightmost value gives the percentage of instances solved to proven optimality.



the profiles, the cumulative distribution function of the performance ratio

$$\rho_s(\tau) = \frac{100}{|P|}\big|\{p \in P : r_{ps} \leq \tau\}\big|$$

is displayed for each setting $s \in S$. In particular, the value $\rho_s(1)$ is the percentage of instances, for which setting s is the fastest, and $\rho_s(r_M - 1)$ gives the percentage of instances setting s manages to solve to optimality. In the performance profile plot, those two values correspond, respectively, to the leftmost and rightmost point of the graph for setting s.

The performance profile plot for all 232 instances from the classes MIPLIB, INTERDICTION and DENEGRE is given in Figure 2. Note that the horizontal axis is log-scaled. Analyzing the obtained results, we may conclude that, consistently over all different instance classes, our intersection cuts embedded in the branch-and-cut framework significantly outperform the BENCHMARK code. Setting SEP-2a turns out to be the best performing one: for 28% of all instances, SEP-2a is the fastest approach, and it manages to solve to optimality 71% of them within the time limit of one hour. In comparison, the BENCHMARK code is the fastest one for only 8% of all instances and manages to solve only 42% of them to optimality.

To evaluate usefulness of ING cuts, we separately considered only binary instances (namely, those from classes INTERDICTION and MIPLIB) and compared our setting ING with the five remaining ones. The corresponding performance profile is reported in Figure 3. We observe that ING outperforms BENCHMARK by a large margin, being the fastest performing approach for about 15% of all binary instances, and solving to optimality 48% of them. On the contrary, BENCHMARK is almost never the fastest one, and it manages to solve only 32% of all binary instances to optimality.

In Table 4 we further summarize the obtained results. We provide the number of solved instances (#) per each class, the *shifted geometric mean* of computing times and the number of branch-and-cut nodes, as well as the average gap. The shifted geometric mean for a given shift s and values $v_1, v_2, \ldots, v_n$ is defined as $\sqrt[n]{\prod(v_i + s)} - s$ (see, e.g., [1]). Computing times and number of nodes are shifted by 10 seconds and 100 nodes, respectively. The percentage gap for each instance is calculated as $\min\{100,\ 100 \cdot (UB - LB)/(|UB| + 10^{-10})\}$, i.e., gaps are clipped to 100% to avoid too-large values that would make the comparison harder.

Figure 3: Performance profile plot over all binary instances (classes `INTERDICTION` and `MIPLIB`). Setting `ING` is now included.

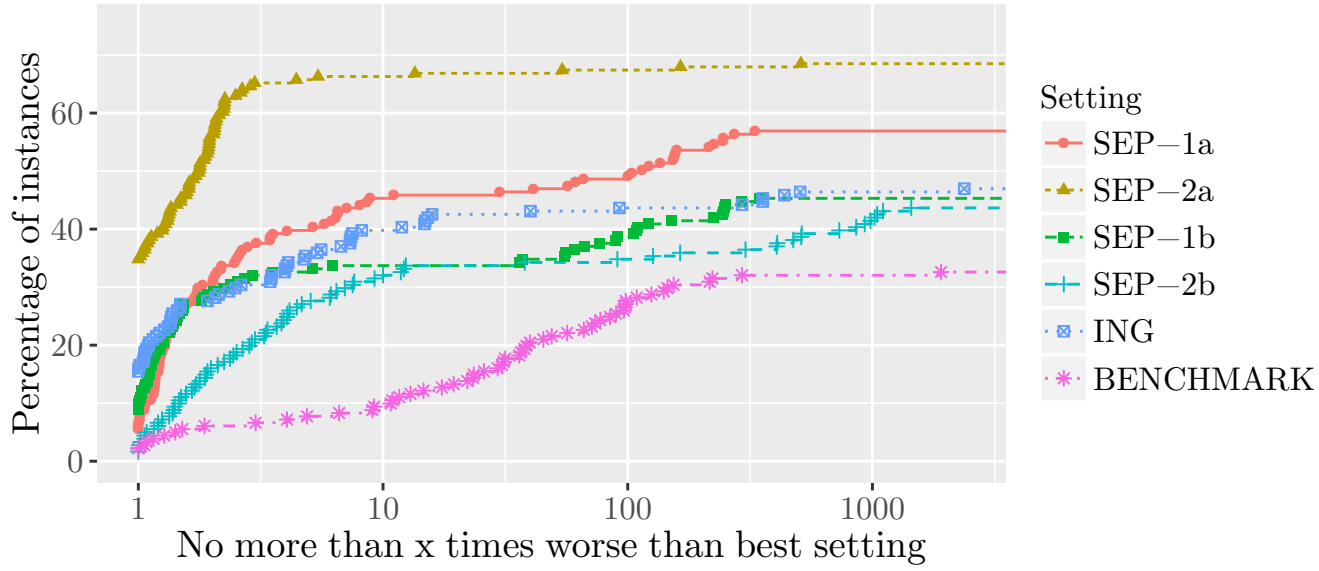


Table 4: Summary of obtained results. We report the number of solved instances (#), the shifted geometric mean for computing time ($t[s]$) and for number of nodes (*nodes*), and the average gaps ($g[\%]$).

	`MIPLIB` (57 inst.s)				`INTERDICTION` (125 inst.s)				`DENEGRE` (50 inst.s)			
setting	#	$t[s]$	*nodes*	$g[\%]$	#	$t[s]$	*nodes*	$g[\%]$	#	$t[s]$	*nodes*	$g[\%]$
`SEP-1a`	20	599	9655.9	27.65	83	148	36769.3	33.06	42	40	574.0	4.61
`SEP-2a`	20	618	8552.9	26.12	104	50	2513.4	4.64	40	60	692.3	7.16
`SEP-1b`	18	660	100475.8	27.85	64	245	240859.4	48.39	45	35	12452.1	3.89
`SEP-2b`	18	666	80172.1	28.42	61	317	295595.8	41.55	43	39	13421.1	5.06
`ING`	16	872	172334.9	30.71	69	173	71967.8	44.35	-	-	-	-
`BENCHMARK`	15	954	234670.7	31.78	44	496	1310639.5	63.45	38	58	27918.5	9.20

Analyzing Table 4, one easily concludes that the most-challenging instances considered in our computational study are those from the `MIPLIB` class. Computing times, number of nodes and final gaps are much larger than the respective values of `DENEGRE` class. The setting `SEP-2a` turns out to be the best performing one for classes `MIPLIB` and `INTERDICTION`. Its success is particularly striking for the `INTERDICTION` class, for which the average gap is only 4.64%, whereas for the remaining five settings this gap remains larger than 30%. This indicates that SEP-2 model for separating ICs has a practical value when the follower has a clean (combinatorial) substructure, which is exploited by our facet-removal scheme for enlarging the bilevel-free polyhedron.

A slightly different behavior can be observed for instances from the `DENEGRE` class. Recall that this class contains very small instances with up to 20 integer variables. Most of these instances are very easy and can be solved within a fraction of a second. See, for example, Table 1, which reports the computing times for `BENCHMARK` over the largest 30 instances from this class. For such small instances, separating intersection cuts at every node of the B&C tree is too time consuming, and it does not pay off in general. This explains why the settings with `max_node_cuts`=0 at the non-root B&C nodes (`SEP-1b`, `SEP-2b` and also `BENCHMARK`) perform much better for these particular cases.

6 Conclusions

We have presented a finitely-convergent (under appropriate conditions) branch-and-cut method for MIBLP, that is intended to be a modification of a classical MILP scheme. By design, our approach focuses on the add-on extras required to convert a branch-and-cut MILP exact code into a valid MIBLP solver. In this way, we inherit the rich set of tools (cuts, heuristics, propagations, etc.) available in modern MILP solvers, and concentrate on bilevel-specific issues.

Valid intersection cuts for MIBLP have been proposed, along with the corresponding separation procedures. We have also introduced a class of "Informed No-Good" (ING) cuts that can be used in the pure-binary case. An extensive computational analysis on different classes of instances from the literature has been reported, showing that the proposed approach outperforms previous proposals by a large margin.

Future work should address the use of different bilevel-free polyhedra for deriving deeper intersection cuts. Different kinds of cuts not requiring the LP basis (like ING cuts) are also of interest and should be investigated as well.

Acknowledgment

This research was funded by the Vienna Science and Technology Fund (WWTF) through project ICT15-014. The work of M. Fischetti and M. Monaci was also supported by the University of Padova (Progetto di Ateneo "Exploiting randomness in Mixed Integer Linear Programming"), and by MiUR, Italy (PRIN project "Mixed-Integer Nonlinear Optimization: Approaches and Applications"). The work of I. Ljubić and M. Sinnl was also supported by the Austrian Research Fund (FWF, Project P 26755-N19). The authors thank M. Caramia and T. Ralphs for providing the instances used in [5] and [6], respectively.

References

[1] T. Achterberg. *Constraint Integer Programming.* PhD thesis, Technische Universität Berlin, Germany, 2009.

[2] C. Audet, J. Haddad, and G. Savard. Disjunctive cuts for continuous linear bilevel programming. *Optimization Letters*, 1(3):259–267, 2007.

[3] E. Balas. Intersection cuts–a new type of cutting planes for integer programming. *Operations Research*, 19(1):19–39, 1971.

[4] R. E. Bixby, S. Ceria, C. M. McZeal, and M. W. P Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. *Optima*, 58:12–15, 1998.

[5] M. Caramia and R. Mari. Enhanced exact algorithms for discrete bilevel linear problems. *Optimization Letters*, 9(7):1447–1468, 2015.

[6] S. DeNegre. *Interdiction and Discrete Bilevel Linear Programming.* PhD thesis, Lehigh University, 2011.

[7] S. DeNegre and T. K. Ralphs. A branch-and-cut algorithm for integer bilevel linear programs. In *Operations research and cyber-infrastructure*, pages 65–78. Springer, 2009.

[8] E. D. Dolan and J.J. Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.

[9] J.J. Dongarra. Performance of various computers using standard linear equations software. 2014.

[10] M. Fischetti, I. Ljubic, M. Monaci, and M. Sinnl. Intersection cuts for bilevel optimization. In *IPCO Proceedings*, LNCS. Springer, 2016.

[11] P.-M. Kleniati and C. S. Adjiman. A generalization of the branch-and-sandwich algorithm: From continuous to mixed-integer nonlinear bilevel problems. *Computers & Chemical Engineering*, 72:373 – 386, 2015.

[12] M. Köppe, M. Queyranne, and C. T. Ryan. Parametric integer programming algorithm for bilevel mixed integer programs. *Journal of Optimization Theory and Applications*, 146(1):137–150, 2010.

[13] A. Lodi, T. K. Ralphs, and G. J. Woeginger. Bilevel programming and the separation problem. *Math. Program.*, 146(1-2):437–458, 2014.

[14] P. Loridan and J. Morgan. Weak via strong Stackelberg problem: new results. *Journal of Global Optimization*, 8:263–297, 1996.

[15] A. Mitsos. Global solution of nonlinear mixed-integer bilevel programs. *Journal of Global Optimization*, 47(4):557–582, 2010.

[16] J. Moore and J. Bard. The mixed integer linear bilevel programming problem. *Operations Research*, 38(5):911–921, 1990.

[17] T. K. Ralphs and E. Adams. Bilevel instance library, 2016. `http://coral.ise.lehigh.edu/data-sets/bilevel-instances/`.

[18] G. K. Saharidis and M. G. Ierapetritou. Resolution method for mixed integer bi-level linear problems based on decomposition technique. *Journal of Global Optimization*, 44(1):29–51, 2009.

[19] P. Xu and L. Wang. An exact algorithm for the bilevel mixed integer linear programming problem under three simplifying assumptions. *Computers & Operations Research*, 41:309–318, 2014.