A dynamic reformulation heuristic for Generalized Interdiction Problems

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

[1]*DEI, University of Padua, Italy.*
[2]*DEI, University of Bologna, Italy.*
[3]*ISOR, University of Vienna, Vienna, Austria.*

Abstract

We consider a subfamily of mixed-integer linear bilevel problems that we call Generalized Interdiction Problems. This class of problems includes, among others, the widely-studied interdiction problems, i.e., zero-sum Stackelberg games where two players (called the leader and the follower) share a set of items, and the leader can interdict the usage of certain items by the follower. Problems of this type can be modeled as Mixed-Integer Nonlinear Programming problems, whose exact solution can be very hard. In this paper we propose a new heuristic scheme based on a single-level and compact mixed-integer linear programming reformulation of the problem obtained by relaxing the integrality of the follower variables. A distinguished feature of our method is that general-purpose mixed-integer cutting planes for the follower problem are exploited, on the fly, to dynamically improve the reformulation. The resulting heuristic algorithm proved very effective on a large number of test instances, often providing an (almost) optimal solution within very short computing times.

1 Introduction

A general bilevel optimization problem is defined as

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

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

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

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.

We will refer to x and y as the *leader* and *follower* decision variable vectors, respectively. Accordingly, $F(x,y)$ and $G(x,y) \leq 0$ denote the leader objective function and constraints, while (3) defines 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 [1].

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

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

[*]matteo.fischetti@unipd.it
[†]michele.monaci@unibo.it
[‡]markus.sinnl@univie.ac.at

one can restate the bilevel optimization problem through the so-called *value-function reformulation* [2] below:

$$\min F(x, y) \tag{5}$$
$$G(x, y) \leq 0 \tag{6}$$
$$g(x, y) \leq 0 \tag{7}$$
$$(x, y) \in \mathbb{R}^n \tag{8}$$
$$f(x, y) \geq \Phi(x). \tag{9}$$

Problem (5)-(8) is usually called the *High-Point Relaxation* (HPR), which is used in many solution approaches to bilevel programming, including [3, 4].

A *Mixed-Integer Bilevel Linear Problem* (MIBLP) is a bilevel problem in which both objective functions $F(x, y)$ and $f(x, y)$ are linear (or affine), and each constraint in $G(x, y) \leq 0$ and in $g(x, y) \leq 0$ is either a linear inequality, or stipulates the integrality of a certain component of (x, y).

Relevant special cases of MIBLPs are known as *interdiction problems* (or interdiction games). They can be seen as two-player zero-sum Stackelberg games [5] where the leader and follower share a set of items, and the leader can select some items and interdict their usage to the follower. The two problems optimize over the same objective function, but in the opposite direction. Connection between the leader and the follower optimization problems is established through binary "interdiction variables" that are controlled by the leader, a leader solution x being sometimes called *interdiction policy* in this context. Interdiction problems model important applications including marketing [3], defense of critical infrastructures [6, 7], and fighting of drug smuggling [8] and illegal nuclear projects [9, 10].

In this paper we consider a generalization of interdiction problems, to be formally described in Section 3, and propose a heuristic solution scheme. In our algorithm, we first relax integrality of the follower variables, obtaining a bilevel problem that can be reformulated into a standard (i.e., single-level and compact) Mixed-Integer Linear Program (MILP). Then we solve the reformulated problem through a black-box MILP solver, and find an optimal solution, say $(\bar{x}, \bar{y})$. As the latter solution is typically not feasible for the original interdiction problem, we solve the original problem after fixing $x = \bar{x}$ (this restricted problem being a MILP itself) to get a hopefully good feasible solution $(\bar{x}, \hat{y})$.

Three variants of our basic heuristic are presented and computationally analyzed on a very large test set with 1200+ instances both from the literature and randomly generated. Although quite simple, even the most basic variant proved successful for a large number of instances, often providing an optimal solution within negligible computing time. For the hardest instances, however, our most sophisticated version based on a dynamic reformulation gives a significant performance improvement.

Our main contributions can be summarized as follows:

- We introduce a generalization of the interdiction problems typically addressed in the literature, obtained by removing the assumption that the leader and follower linear objective functions are one the opposite of the other. Generalized interdiction problems cannot be casted as min-max or max-min problems, so many classical results that hold for standard interdiction problems need to be generalized to the new setting.

- For such generalized interdiction problems, we present a new MILP reformulation of the problem obtained by relaxing integrality conditions of the follower variables. Similar reformulations have been addressed in the literature for standard interdiction problems both with a MILP follower (e.g., in [11, 12, 13]) and with an LP follower (in this latter case, the reformulation is equivalent to the original problem; see, e.g., [14, 15, 16]). Moreover, for another class of min-max problems, namely the min-max regret robust problems, a related relax-dualise-reformulate idea has been used to devise heuristics (e.g., in [17, 18]). As far as we know, however, the min-max reformulations from the literature were never extended to the generalized interdiction setting we consider in the present paper.

- We describe two basic interdiction heuristics in the spirit of [11, 13], and propose an improved version based on a dynamic reformulation. This latter algorithm iteratively generates, on the fly, valid cutting

planes based on the integrality of the follower variables, thus producing improved reformulations and better solutions. The resulting approach can be viewed as a (row and) column generation method applied to the reformulation, whose effectiveness has been confirmed by extensive computational tests. To the best of our knowledge, a similar mechanism was never used in the context of general-purpose interdiction or min-max heuristics.

- Our heuristic solution scheme can easily be adapted to more general min-max problems, and also to the setting addressed in [14] where interdiction penalties (as opposed to constraints) are considered.
- We report very extensive computational tests on more than 1200 instances both from the literature and randomly generated. This is, by far, the most extensive study on general-purpose interdiction heuristics reported in the literature.

The paper is organized as follows. Previous work on interdiction problems and heuristic methods for bilevel (integer) linear programming problems is briefly surveyed in Section 2. The generalized interdiction problem we address in our work is mathematically stated in Section 3, while a single-level compact reformulation of the problem without integrality requirement on the y variables is introduced in Section 4. Section 5 describes our basic heuristic along with two extensions intended to produce improved solutions. Extensive computational results on various classes of test instances are reported in Section 6, while Section 7 draws some conclusions and addresses future directions of work.

2 Previous work

We next focus on previous work regarding interdiction problems and heuristic approaches for bilevel (integer) linear programming; for previous contributions on exact approaches for MIBLP we refer the reader to, e.g., [4].

Exact Solvers for Interdiction Problems When the follower problem in an interdiction problem can be formulated as linear program, duality-based reformulations give exact algorithms. This idea has been used, e.g., in [16] for the maximum flow problem, in [15] for multi-commodity flow problems, and in [19] for the spanning tree problems. The shortest-path variant for a problem where interdiction does not forbid the use of an item but causes a penalty in the follower objective is studied in [14], where a duality-based exact algorithm is given. Finally, for another variant where interdiction is not a discrete decision, but continuously increases a follower penalty in the objective, duality has also been used in the design of algorithms, see, e.g., [20].

A basic interdiction problem with discrete follower is the Knapsack Interdiction Problem (KIP) studied in [3, 13, 21]. The problem consists of a Stackelberg game where both the leader and the follower players fill their private knapsacks by choosing items from a common set N. In the first stage, the leader chooses her items subject to her own knapsack capacity (called interdiction budget). In the second step, the follower solves a 0/1 knapsack problem and selects some of the items that are not taken by the leader, with the aim of maximizing the profit of the collected items. The goal of the leader is to obtain the worst possible outcome for the follower. A typical application of this problem arises in marketing, when a company dominates the market and another company wishes to design a marketing campaign, while choosing the specific geographic regions to target subject to the available budget.

KIP is exactly reformulated in [3] as a single-level MILP with an exponential number of constraints, to be separated on the fly by using disjunctive cut-generating LPs. In [13], an iterative MILP-based procedure is presented, where lower and upper bounds are sequentially improved until a provably optimal solution is reached. In this latter approach, upper bounds are computed by solving a heuristic single-level MILP reformulation akin to the one we use in the present paper, but specialized for KIPs.

In a more general setting, interdiction problems play a very important role in the applications arising in the so-called attacker-defender games, where two players have conflicting objectives and share a set of resources. In particular, interdiction problems on networks have beed widely studied in the recent literature;

see, e.g., the shortest-path interdiction problem given in [14], and the survey on network interdiction given in [22]. Complexity results for some special cases of these problems are given in [23].

Three ideas for deriving a generic solver for interdiction games have been proposed in [21]. Very recently, an exact branch-and-cut solution scheme for interdiction problems with monotonicity assumptions on the follower problem has been proposed in [24], which is based on a Benders-like reformulation that is enhanced by additional classes of cuts.

Heuristics for Interdiction Problems Problem-specific heuristic approaches have been proposed in the literature, e.g., [25] presents heuristics for multi-stage interdiction of stochastic networks, while [11] describes a reformulation heuristic for an interdiction problem arising in the design of a new branching strategy for MILP solvers. However, to the best of our knowledge, the only generic heuristic algorithm for interdiction problems is the greedy heuristic proposed in [3]. In this algorithm, an interdiction policy is greedily constructed by iteratively picking items with the largest follower coefficient in the objective, while satisfying the interdiction budget. A drawback of this approach is that it is "blind" with respect to the structure of the follower problem.

Heuristics for Bilevel Problems For bilevel (integer) linear programming problems, some heuristic schemes exist, depending on varying assumptions on the problem structure.

The thesis [3] presents four heuristic approaches (in addition to the above-mentioned greedy approach for interdiction problems). Observe that, in the setting studied in [3], the leader constraints are only of the form $G(x) \leq 0$, i.e., no y-variable appears in the leader constraints. Let $(\hat{x}, \hat{y})$ be a bilevel feasible solution. The first heuristics consists of adding the cut $f(x, y) \leq \phi(\hat{x})$ to the HPR, and solving the resulting model. The rationale is that there may be multiple optimal follower solutions for the given $\hat{x}$, and adding the constraint above to HPR may produce a solution with a better leader objective. Of course, the produced solution may change the value of the x variables, in which case an additional check for bilevel feasibility is needed. The second heuristic optimizes the follower objective function over the HPR feasible set. This produces a solution that is guaranteed to be bilevel feasible, though likely of bad quality, since the leader objective is ignored. Thus, a cut-off constraint for the leader objective based on the value of incumbent solution is added to the formulation. The addition of this cut may lead to bilevel infeasible solutions, thus in this case too a feasibility check must be executed. The third heuristic is based on multi-objective programming and computes, by using a weighted-sum method, Pareto optimal solutions for the bi-objective problem with both leader and follower objective functions as objectives. These solutions are then checked for bilevel feasibility and the best feasible one is taken. Finally, the last heuristic is a ping-pong scheme that iteratively fixes the variables in one of the two problems, and solves the other, until a bilevel feasible solution is found.

We observe that Bilevel Linear Programming (BLP) can be reformulated as a nonlinear single-level problem using KKT-conditions. Based on this reformulation, [26] proposed a hybrid tabu-ascent algorithm that works in three phases. First an initial solution is produced, then a local ascent phase is executed, and finally, a tabu search phase is applied to improve the current solution. Note that this reformulation has also been used in exact approaches for BLP, see, e.g., [27].

As to metaheuristics for BLP, we mention the tabu search proposed in [28] that can be used for bilevel linear problems in which the leader is mixed-integer, and the genetic algorithm in [29]. We refer the reader to the book [30] for further details on metaheuristics for BLPs.

3 The problem

As already mentioned, in the present paper we focus on a special case of MIBLP that we call *Generalized Interdiction Problem* (GIP), whose value-function reformulation reads:

$$
\begin{aligned}
(GIP) \quad & \min c_x^T x + c_y^T y && && (10)\\
& G_x x + G_y y \le G_0 && && (11)\\
& By \le b \text{ and } y \ge 0 && && (12)\\
& y_j \le UB_j\,(1 - x_j), && \forall j \in N && (13)\\
& d^T y \ge \Phi(x) && && (14)\\
& x_j \in \{0,1\}, && \forall j \in N && (15)\\
& x_j \text{ integer}, && \forall j \in J_x && (16)\\
& y_j \text{ integer}, && \forall j \in J_y. && (17)
\end{aligned}
$$

In the above model, the variable set is partitioned into sets N_x and N_y that contain the leader and follower variables, respectively, while sets $J_x \subseteq N_x$ and $J_y \subseteq N_y$ identify the indices of the integer-constrained leader and follower variables, respectively. Finally, c_x, c_y, G_x, G_y, G_0, B, b, UB and d denote given rational matrices/vectors of appropriate size.

Set $N \subseteq N_x \cap N_y$ identifies the *items* that can be "interdicted" by the leader: according to the interdiction constraints (13), this happens when the leader chooses $x_j = 1$, which forces the follower to choose $y_j = 0$, whereas in case $x_j = 0$ the follower is free to choose $y_j \in [0, UB_j]$.

To simplify notation, we assume that constant upper bounds on the y variables (including $y_j \le UB_j$ for all $j \in N$) are stated explicitly and belong to system $By \le b$.

The value function $\Phi(x)$ in (14) is computed, for a given x, by solving the *follower MILP*:

$$
\begin{aligned}
\Phi(x) := \max\, & d^T y && && (18)\\
& By \le b \text{ and } y \ge 0 && && (19)\\
& y_j \le UB_j\,(1 - x_j), && \forall j \in N && (20)\\
& y_j \text{ integer}, && \forall j \in J_y. && (21)
\end{aligned}
$$

Thus, unlike general MIBLPs, in the GIP framework the only allowed follower constraints depending on x are of the form (20).

As customary, we assume that the follower MILP is bounded and feasible for any given x of interest.

In what follows we will rephrase the interdiction constraints (20) in the follower problem through the bilinear conditions $x_j\, y_j = 0$ for all $j \in N$. As x_j and y_j are both nonnegative, we can relax these latter equations in a Lagrangian way through very large positive multipliers M_j (say), and obtain the penalized objective function

$$
d^T y - \sum_{j \in N} M_j x_j y_j.
$$

This leads to the following *exact* reformulation for the follower MILP (18)–(21), that we will use throughout:

$$
\begin{aligned}
\Phi(x) := \max\, & d(x)^T y && && (22)\\
& By \le b \text{ and } y \ge 0 && && (23)\\
& y_j \text{ integer}, && \forall j \in J_y && (24)
\end{aligned}
$$

where

$$
d_j(x) := \begin{cases} d_j - M_j x_j, & \text{if } j \in N \\ d_j, & \text{otherwise} \end{cases} \qquad \forall j \in N_y \qquad (25)
$$

As already mentioned, a relevant GIP special case is the widely-studied *Standard Interdiction Problem* (SIP), arising when $G_y = 0$, $c_x = 0$, and $c_y = d$. In this case, the leader and follower subproblems optimize

the same objective function, but in opposite directions, the follower variables are not present in the leader constraints, while the leader variables appear only in the interdiction constraints (20) in the follower. This lead to the following min-max formulation:

$$(SIP) \quad \min_x \max_y \; d(x)^T y \tag{26}$$

$$G_x x \le G_0 \tag{27}$$

$$By \le b \text{ and } y \ge 0 \tag{28}$$

$$x_j \in \{0,1\}, \qquad \forall j \in N \tag{29}$$

$$x_j \text{ integer}, \qquad \forall j \in J_x \tag{30}$$

$$y_j \text{ integer}, \qquad \forall j \in J_y. \tag{31}$$

4 Single-level MILP reformulation

As already stated, the first step of our heuristic consists in relaxing the integrality condition on the y variables everywhere, namely removing constraints (17) and (24) from (10)-(17) and (22)-(24), respectively. Let $\overline{GIP}$ denote the resulting problem, and let $\overline{SIP}$ be obtained in a similar way from the SIP model (26)-(31) by dropping (31).

Note that $\overline{GIP}$ is not a relaxation nor a restriction of the original GIP problem, in that the integrality condition on the y variables is relaxed in both the leader and in the follower problems. Indeed, while relaxing the integrality requirements (17) alone would obviously produce a relaxation, the removal of the follower integrality constraints (24) produces an upper bound $\overline{\Phi}(x)$ on $\Phi(x)$ and then makes constraint (14) more restrictive. This means that the optimal value of $\overline{GIP}$ can be strictly larger or strictly smaller than the optimal value of GIP. As a matter of fact, $\overline{GIP}$ turns out to be a restriction of GIP in case the integrality requirements (17) in the leader are redundant—as it happens, in particular, for the standard interdiction case SIP. It is instead a relaxation of GIP in case the integrality condition (24) is redundant for the follower. If both cases arise, $\overline{GIP}$ is just equivalent to the original problem GIP; this happens, in particular, when $J_y = \emptyset$.

We next show how Linear Programming (LP) duality can be used to restate $\overline{GIP}$ and $\overline{SIP}$ as single-level MILPs. To this end, for a fixed x, let

$$\overline{\Phi}(x) := \max\{d(x)^T y : By \le b, y \ge 0\} \tag{32}$$

denote the optimal value of the LP relaxation of the follower problem (that we assume be feasible and bounded), with $\overline{\Phi}(x) \ge \Phi(x)$. By using standard LP duality, we get

$$\overline{\Phi}(x) = \min\{u^T b \,:\, u^T B \ge d(x)^T,\, u \ge 0\}. \tag{33}$$

We address $\overline{SIP}$ first, for which the reformulation is in fact quite natural and can be found, e.g., in [14, 22, 11, 13].

Theorem 4.1. *Model $\overline{SIP}$ can be restated as*

$$(\overline{SIP}) \quad \min u^T b \tag{34}$$

$$G_x x \le G_0 \tag{35}$$

$$x_j \in \{0,1\}, \qquad \forall j \in N \tag{36}$$

$$x_j \text{ integer}, \qquad \forall j \in J_x \tag{37}$$

$$u^T B \ge d(x)^T \text{ and } u \ge 0. \tag{38}$$

Proof. Just observe that (34)-(38) can be obtained from (26)-(31) by applying standard LP duality to the LP relaxation of the inner maximization problem, thus removing the y variables from the model. □

As to $\overline{GIP}$, y variables cannot be projected away due to the presence of the leader constraints (11), so the model involves explicit LP optimality conditions.

Theorem 4.2. *Model $\overline{GIP}$ can be restated as*

$$
\begin{aligned}
(\overline{GIP}) \quad & \min c_x^T x + c_y^T y && && (39)\\
& G_x x + G_y y \leq G_0 && && (40)\\
& By \leq b \text{ and } y \geq 0 && && (41)\\
& y_j \leq U B_j\,(1 - x_j), && \forall j \in N && (42)\\
& x_j \in \{0,1\}, && \forall j \in N && (43)\\
& x_j \text{ integer}, && \forall j \in J_x && (44)\\
& u^T B \geq d(x)^T \text{ and } u \geq 0 && && (45)\\
& d^T y \geq u^T b. && && (46)
\end{aligned}
$$

Proof. For any given x, primal feasibility of y for the follower LP relaxation (32) comes from (41), while (45) expresses feasibility of u for its dual (33). According to the strong duality theorem, the pair (y, u) is then optimal if and only if $d(x)^T y = u^T b$, where $=$ can be replaced by $\geq$ because of weak duality. In addition, (42)-(43) ensure that $d(x)^T y = d^T y$, hence in (46) one is allowed to replace the bilinear condition $d(x)^T y \geq u^T b$ by the equivalent linear constraint $d^T y \geq u^T b$. Finally, observe that the presence of $d(x)$ is instead not problematic in (45) as it does not affect its linearity. The claim follows. □

It is worth noting that each constraint in (45) associated with an item $j \in N$ has the form

$$d_j - u^T B_j \leq M_j x_j,$$

hence it involves a big-M coefficient that can create troubles to the MILP solver. By exploiting the fact that x_j is a binary variable, however, in the relevant case $B_j \geq 0$ one has $u^T B_j \geq 0$ and coefficient M_j can safely be replaced by d_j. These (and similar) coefficient strengthenings are automatically applied by modern MILP solvers and typically produce preprocessed MILP models that, according to our computational experience, are not too hard to solve.

5 Heuristics

We next describe three heuristics based on the solution of the single-level MILP reformulation ($\overline{GIP}$) or ($\overline{SIP}$) described in the previous section.

Our first heuristic, `ONE-SHOT`, is straightforward to implement, and proved very fast and effective for many cases in our testbed. For the hardest cases, improved solutions can be obtained by implementing a more advanced solution scheme. We describe two such schemes.

The first scheme (`ITERATE`) is also quite easy to implement, and just iterates the application of the `ONE-SHOT` heuristic so as to produce diversified solutions.

The second scheme (`DYN-REF`) requires a more advanced implementation, as valid cuts exploiting the integrality of the y's in the follower MILP are dynamically generated and added to the reformulation solved by `ITERATE`.

The first two heuristics (`ONE-SHOT` and `ITERATE`) can be viewed as a generalization of the solution scheme described in [11] for KIPs, whereas the third one (`DYN-REF`) is a new dynamic reformulation heuristic that makes use, for the first time, of the general-purpose mixed-integer cuts that are automatically generated by a modern branch-and-cut solver when solving the follower MILP.

Algorithm 1: The refining procedure `REFINE`($\bar{x}$)

Input : A leader solution $\bar{x}$;
Output: A heuristic GIP solution $(\bar{x}, \bar{y})$;

1 Solve the follower MILP (22)-(24) for $x = \bar{x}$ to compute $\bar{\varphi} := \Phi(\bar{x})$;
2 Restrict the GIP model (10)-(17) by fixing $x = \bar{x}$ and by replacing the nonlinear inequality (14) with the linear constraint $d^T y \geq \bar{\varphi}$;
3 Solve the resulting MILP, and let $(\bar{x}, \bar{y})$ be the optimal solution found;
4 **return** $(\bar{x}, \bar{y})$;

5.1 The `ONE-SHOT` heuristic

Our basic heuristic scheme is described in Algorithms 1 and 2.

Algorithm 1 describes the *refining procedure* given, e.g., in [31], that computes a complete feasible GIP solution $(\bar{x}, \bar{y})$ starting from a leader vector $\bar{x}$. It requires the solution of two single-level MILPs: the follower MILP for $x = \bar{x}$ at Step 1 to compute $\Phi(\bar{x})$, and a linearization of the GIP model (10)-(17) based on the fact that $\Phi(x)$ becomes a constant when x is fixed (Steps 2–3). Algorithm 1 typically requires short computing time, as both MILPs are much easier than the original nonlinear GIP model. Note that leader variables x_j not appearing in the follower MILP (22)-(24) (if any), do not affect the value $\bar{\varphi} = \Phi(\bar{x})$ computed at Step 1, hence they need not to be fixed at Step 2.

Algorithm 1 can be simplified for standard interdiction problems. Indeed, in that case Step 3 can be omitted, as the optimal follower solution $\bar{y}$ computed at Step 1 can directly be appended to the input vector $\bar{x}$ to obtain the final solution $(\bar{x}, \bar{y})$ to be returned.

It is worth noting that, for GIPs, the solution $(\bar{x}, \bar{y})$ returned by `REFINE`, besides being feasible, can have an improved cost with respect to the (partial) solution provided on input, hence the name of procedure.

Algorithm 2 is used to find a reasonable leader vector, say $\bar{x}$, to feed Algorithm 1. This is obtained by relaxing the integrality of the y variables to be able to apply the single-level MILP reformulation described in the previous section. The reformulated MILP is typically not too hard to solve, but for very large instances its exact solution can be too demanding, thus a time limit can be imposed.

Algorithm 2: Our basic `ONE-SHOT` heuristic scheme

Input : The GIP model (10)-(17);
Output: A heuristic GIP solution $(\bar{x}, \bar{y})$;

1 Relax the integrality of the y variables, namely constraints (17) and (24);
2 Restate the resulting problem as $(\overline{GIP})$;
3 Solve the resulting single-level MILP (possibly with a time limit), and let $(\bar{x}, \cdot)$ be the optimal (or best) solution found;
4 $(\bar{x}, \bar{y})$:= `REFINE`($\bar{x}$);
5 **return** $(\bar{x}, \bar{y})$;

Needless to say, if the input problem is a SIP, the associated reformulation defined at Step 2 is $(\overline{SIP})$. In addition, if at any step of both algorithms the MILP at hand has no feasible solution (or no one could be found in the given time limit), a void solution $(\bar{x}, \bar{y})$ of cost $+\infty$ is returned.

5.2 The `ITERATE` heuristic

Our `ITERATE` heuristic is described in Algorithm 3. At Step 2, a single-level MILP reformulation is defined; this is either $(\overline{GIP})$ or $(\overline{SIP})$, depending on the input model (10)-(17). At each iteration, the single level reformulation is solved with a time limit, to collect a number of almost-optimal solutions (Step 5). These

solutions coincide, e.g., with the various incumbents by the MILP solver. The refining procedure `REFINE` is then applied to each such solution (Step 7). Each time a new heuristic GIP solution (x^k, y^k) is available, at Step 9 we possibly update the incumbent (x^*, y^*). When all the available solutions have been refined, the whole procedure is repeated after adding to the GIP model (10)-(17) a no-good constraint

$$\sum_{j \in N: x_j^k = 0} x_j + \sum_{j \in N: x_j^k = 1} (1 - x_j) \geq 1 \tag{47}$$

for each generated leader solution x^k, thus preventing these solutions to be considered again in subsequent iterations.

As to the iterated call to procedure `REFINE` at Step 7, each execution typically requires very short computing time. Moreover, for SIP one can abort the whole refining procedure for a given x^k as soon as a follower solution of value greater than or equal to $z^* = d^T y^*$ is encountered. Indeed, for the follower MILP being a maximization problem, the subsequent part of the `REFINE` run can only increase the incumbent cost of the follower, whose best solution, say y^k, will produce a heuristic solution (x^k, y^k) of cost $d^T y^k \geq z^*$, meaning that the internal incumbent solution (x^*, y^*) will not be updated.

Algorithm 3: The `ITERATE` heuristic

Input : The GIP model (10)-(17) and a time limit `TL`;
Output: A heuristic GIP solution (x^*, y^*);

1 Initialize (x^*, y^*) with a dummy solution of cost $z^* := +\infty$;
2 Build the single-level MILP reformulation $(\overline{GIP})$;
3 **while** *time limit* `TL` *is not exceeded* **do**
4 Let `rt` be the time remaining to reach the time limit;
5 Solve the current single-level MILP reformulation with a time limit of 0.95 `rt`, and let $(x^1, \cdot), \ldots, (x^K, \cdot)$ be the collection of solutions found;
6 **for** $k = 1, \ldots, K$ **do**
7 $(x^k, y^k) :=$ `REFINE`(x^k);
8 **if** $c_x^T x^k + c_y^T y^k < z^*$ **then**
9 set $z^* := c_x^T x^k + c_y^T y^k$ and $(x^*, y^*) := (x^k, y^k)$;
10 **end**
11 Add a no-good constraint (47) for x^k to the current single-level MILP reformulation;
12 **end**
13 **end**
14 **return** (x^*, y^*);

We finally observe that, if a sufficiently large time limit is allowed, the `ITERATE` heuristic turns out to be an exact approach for GIP. In this respect, the approach is similar to the one proposed in [13] for IKP, where specific cuts exploiting the structure of the problem are used instead of the fully general no-good constraints (47).

5.3 The `DYN-REF` heuristic

`DYN-REF` is our new dynamic reformulation heuristic. It is based on the observation that the smaller the follower integrality gap, the better the single-level MILP reformulation $(\overline{GIP})$ or $(\overline{SIP})$ approximates the original model (GIP) or (SIP), respectively. Our new algorithm then implements an advanced version of `ITERATE` in which the *follower* MILP formulation (22)-(24) is iteratively strengthened by adding valid inequalities that exploit the integrality of the y variables in the follower problem.

To this end, observe that the constraints in our follower MILP formulation (22)-(24) do not depend on x. Therefore, after each execution of Step 1 within the refining Algorithm 1, one can just collect the root-node

cuts generated by the MILP solver, and add them to the follower system $By \leq b$. The resulting single-level MILP reformulation solved at Step 5 of Algorithm 3 is thus dynamically updated, and new dual variables are generated for the newly added rows of (B, b).

Note that, for SIP, the resulting scheme can be interpreted as a dynamic column-generation method where new columns of B^T are added to the $\overline{SIP}$ reformulation in (38) (to be re-written as $B^T u \geq d(x)$), while for GIP we have a row-and-column generation scheme as new rows are also added in (41). A peculiarity of our scheme is that the generation of the new columns of B^T does not require an ad-hoc pricing algorithm, in the sense that we just re-use the cutting planes generated by the MILP solver invoked with function `REFINE`.

The generated cuts, besides speeding-up the solution of the next follower MILPs, are very important in that they hopefully reduce the follower integrality gap and produce a tighter $\overline{GIP}/\overline{SIP}$ approximation. In a sense, our approach is gathering local information obtained by solving the follower MILPs for different objective functions $d(x^1)^T y, \cdots, d(x^K)^T y$ within function `REFINE`, and iteratively incorporates this information in the current reformulation to let the heuristic better cope with the consequences of y-integrality at the follower level.

In this way, we expect a better approximation (hence, hopefully better solutions) at each execution of the while-loop of Algorithm 3, in particular in the hard cases where the follower formulation has a large integrality gap. This behavior is in fact confirmed by the computational results reported in the next section.

6 Computational results

In this section we report the outcome of the extensive computational analysis that we performed on a very large testbed with 1286 instances. Computing times refer to four-thread runs on a quad-core Intel Xeon E3-1220V2 @3.1 GHz computer with 12GB of RAM.

6.1 Implementation

Our heuristics have been implemented in C, using the commercial solver IBM ILOG CPLEX 12.6.3 as underlying branch-and-cut framework.

To enhance the performance of Cplex as a heuristic when solving the $\overline{GIP}$ or $\overline{SIP}$ reformulation, in our implementation we switch to the POLISHING [32] heuristic mode after 50% of the imposed time limit. As to the `ITERATE` heuristic and its improved version `DYN-REF`, the collection of K feasible solutions for the current $\overline{GIP}$ (or $\overline{SIP}$) reformulation used at Step 5 of Algorithm 3 corresponds to the solutions that are present in the solution pool of our MILP solver, which contains the various incumbents found during enumeration. In some versions of our codes, we use the POPULATE feature of Cplex to increase the number of feasible solutions of the reformulation without a very significant increase of the required computing time. To avoid overtuning, all the other CPLEX parameters (including those related to cut generation) are left at their default values.

The detailed configurations of the proposed heuristics tested in the computational study are as follows; the "+" sign in the name of a solver means that the POPULATE feature is active.

- `ONE-SHOT` (`OS`): a straightforward implementation of Algorithm 2;
- `ONE-SHOT+` (`OS+`): same as `ONE-SHOT`, but several solutions of the reformulation are generated by using POPULATE (and then refined);
- `ITERATE` (`I`): a straightforward implementation of Algorithm 3;
- `ITERATE+` (`I+`): same as `ITERATE`, but several solutions of the reformulation are generated through POPULATE (and then refined) at each iteration of the while loop;
- `DYN-REF+` (`DR+`): our dynamic reformulation algorithm. This heuristic has the same setting as `ITERATE+`, but the Cplex's root cuts generated during the REFINE procedure are collected and added to the follower model (and hence to the reformulation through additional dual variables); to avoid

overloading the model, cuts are collected only when the best solution of the current reformulation, say $(x^1, \cdot)$, is refined.

We also implemented the only general-purpose interdiction heuristic that we could find in the literature, namely, the greedy algorithm `GREEDY` (`GD`) proposed in [3]. This algorithm was designed for standard interdiction instances and consists of greedily building an interdiction policy by sorting the items according to nonincreasing d_j's (recall that the follower is a maximization problem), and iteratively picking them provided that the leader constraint $G_x x \leq G_0$ is fulfilled. For a more fair comparison, we also implemented a simple 4-thread randomized variant `GRASP` (`GR`) where alternative interdiction policies are repeatedly created by following the recipe of `GREEDY`, but the item which is going to be selected at each iteration is skipped with a probability of 20%, until the given time limit is reached. In the first iteration of `GRASP`, we do not do any skipping, i.e., the solutions of `GRASP` are always at least as good as those of `GREEDY`.

6.2 Testbed

In our testbed, we included both standard interdiction problems and generalized interdiction problems. As to the former, we considered instances from literature, as well as larger versions of some of those instances, randomly generated following the procedures described in the literature. In particular, the interdiction problems considered are presented in Section 6.2.1, and consist of the Knapsack Interdiction Problem (KIP), Multidimensional Knapsack Interdiction Problem (MKIP), Clique Interdiction Problem (CIP) and the Firefighter Problem (FFP). Our testbed does not include instead any instance coming, e.g., from network-flow or shortest-path interdiction problems, as those problems have a continuous follower problem that would make the MILP reformulation (and hence our heuristic) exact.

Table 1: Our testbed. Column #inst reports the total number of instances in each class. All instances not of type GIP are standard interdiction instances ($G_y = 0$, $c_x = 0$, and $c_y = d$). For instances of type KIP and MKIP, there is a one-to-one correspondence between leader and follower variables, while for the other problems, there are also additional variables in the follower.

Class	Source	Type	#inst	$\lvert N_x \rvert$	$\lvert N_y \rvert$	$\lvert N \rvert$
`CCLW`	[13]	KIP	50	35-55	35-55	35-55
`TRSK`	[21]	KIP	150	20-30	20-30	20-30
`D`	[3], [33]	KIP	160	10-50	10-50	10-50
`LKIP`	this paper	KIP	400	100-500	100-500	100-500
`SAC`	[24]	MKIP	144	10-105	10-105	10-105
`TRSC`	[21]	CIP	80	19-94	27-109	19-94
`TRSC+`	this paper	CIP	80	19-94	27-109	19-94
`LCIP`	this paper	CIP	60	546-1593	586-1653	546-1593
`FIRE`	[34]	FFP	72	25-80	50-160	25-80
`GEN`	this paper	GIP	90	20-30	20-30	10-30

To the best of our knowledge, there is no previous computational work for generalized interdiction problems, so we randomly generated a set of GIP instances (denoted by `GEN`) by using the procedure described in Section 6.2.2.

Table 1 gives an overview of the instances in our testbed and reports, for each class, the source in the literature, the associated type of problem, the number of instances and their size. All instances are available online at
`http://homepage.univie.ac.at/markus.sinnl/program-codes/bilevel/`.

6.2.1 Standard Interdiction Instances

Knapsack and Multidimensional Knapsack Interdiction Problem The knapsack interdiction problem is defined as follows

$$\min_{x \in \{0,1\}^{|N|}} \max_{y \in \{0,1\}^{|N|}} \Big\{ d^T y : a^T x \leq a_0, q^T y \leq q_0, x_j + y_j \leq 1, \forall j \in N \Big\},$$

where N denotes the set of items.

There are three sets of KIP instances in literature: instance set `CCLW` from [13], instance set `TRSK` from [21], and instance set `D` from [3] (the latter being derived from multi-objective knapsack instances).

Computational experiments in [24] showed that set `CCLW` contains the most difficult instances; thus, we used the same random procedure to create larger random KIP instances denoted by `LKIP`. The follower data has been created by using the knapsack instance generator of [35], which allows for generation of the profit d_j and weight q_j of each item $j \in N$ in the following nine ways, where u.r. stands for uniformly random:

1. *Uncorrelated*: q_j u.r. in $[1, \overline{R}]$, d_j u.r. in $[1, \overline{R}]$.
2. *Weakly correlated*: q_j u.r. in $[1, \overline{R}]$, d_j u.r. in $[q_j - \overline{R}/10, q_j + \overline{R}/10]$ so that $d_j \geq 1$.
3. *Strongly correlated*: q_j u.r. in $[1, \overline{R}]$, $d_j = q_j + \overline{R}/10$.
4. *Inverse strongly correlated*: d_j u.r. in $[1, \overline{R}]$, $q_j = d_j + \overline{R}/10$.
5. *Almost strongly correlated*: q_j u.r. in $[1, \overline{R}]$, d_j u.r. in $[q_j + \overline{R}/10 - \overline{R}/500, q_j + \overline{R}/10 + \overline{R}/500]$.
6. *Subset-sum*: q_j u.r. in $[1, \overline{R}]$, $d_j = q_j$.
7. *Even-odd subset-sum*: q_j even value u.r. in $[1, \overline{R}]$, $d_j = q_j$.
8. *Even-odd strongly correlated:* q_j even value u.r. in $[1, \overline{R}]$, $d_j = q_j + \overline{R}/10$.
9. *Uncorrelated with similar weights:* q_j u.r. in $[100\,\overline{R}, 100\,\overline{R} + \overline{R}/10]$, d_j u.r. in $[1, \overline{R}]$.

In [13], the authors used $\overline{R} = 100$ and generated instances of type 1 only. The follower budget is set to $q_0 = \lceil \frac{INS}{11} \sum_{j \in N} q_j \rceil$, where INS is the number of the instance, with $1 \leq INS \leq 10$. The leader coefficients a_i are integers chosen u.r. in $[0, \overline{R}]$, while the leader budget a_0 is taken from $[q_0 - 10, q_0 + 10]$. All instances from [13] include at most 55 items. Using the same instance generator, we created larger instances with $|N| \in \{100, 200, 300, 400, 500\}$. For each type of instances and number of items, we randomly generated 10 instances. However, instances of type 9 turned out to be trivial as the leader budget allows for the interdiction of all items, so they have been omitted. Thus, our `LKIP` testbed includes 400 instances.

MKIP is a generalization of KIP in which the leader and/or the follower have several knapsack constraints. For MKIP, a benchmark of 144 instances (set `SAC`) has been proposed in [24], starting from the 0/1 multidimensional knapsack instances included in the SAC-94 library [36]. In particular, the MKIP instances were obtained by (i) taking the first constraint as leader constraint, and the remaining ones as follower constraints; or (ii) considering first half of the constraints as leader constraints, and the remaining ones as follower constraints; or (iii) considering all but the last constraint as leader constraint.

Clique Interdiction Problem Let $G = (V, E)$ be an undirected graph. In CIP, the follower solves a maximum clique problem, and the leader interdicts edges to minimize the size of the maximum clique. Let x be the leader interdiction variables associated with the edges, and y^V and y^E denote the follower binary variables associated with the nodes and the edges, respectively. The clique interdiction problem can defined using the following *extended formulation* [21]:

$$\min_{x \in \{0,1\}^{|E|}} \max_{y \in \{0,1\}^{|V|+|E|}} \sum_{i \in V} y_i^V \tag{48}$$

$$\sum_{(i,j) \in E} x_{ij} \leq k, \tag{49}$$

$$y_i^V + y_j^V \leq 1, \qquad \forall (i,j) \notin E \tag{50}$$

$$y_i^V + y_j^V - y_{ij}^E \leq 1, \qquad \forall (i,j) \in E \tag{51}$$

$$x_{ij} + y_{ij}^E \leq 1, \qquad \forall (i,j) \in E \tag{52}$$

Set TRSC has been introduced in [21] and is generated in the following way. Graphs are constructed for graph densities $d \in \{0.7, 0.9\}$ which yields $|E| = \lfloor d\ |V|\ (|V| - 1)/2 \rfloor$. Each potential edge has equal probability of being created. The interdiction budget k (i.e., the number of edges that can be interdicted) is chosen as $\lceil |E|/4 \rceil$. Ten instances for each d and $|V| \in \{8, 10, 12, 15\}$ have been created, leading to 80 instances.

To study the influence of the problem formulation on the performance of our heuristics, we also generated a new set of instances, denoted by TRSC+, that is obtained from TRSC by adding the following redundant constraints to (48)-(52):

$$y_i^V + y_j^V + y_k^V - y_{ij}^E \leq 1, \qquad \forall i, j, k : \{i, j\} \in E, \{i, k\} \notin E, \{j, k\} \notin E. \tag{53}$$

These constraints are $\{0, 1/2\}$-Chvátal-Gomory cuts [37] obtained by combining inequalities (50)-(51), and are used to strengthen the LP relaxation of the follower formulation which is known to be quite poor.

Moreover, an additional set of larger CIP instances, named LCIP, was generated by using the same procedure as in TRSK (without the redundant constraints (53)) with $|V| \in \{40, 50, 60\}$, thus producing 60 new instances.

Firefighter Problem These instances derive from a trilevel version (following the defender-attacker-defender scheme of [7]) of the critical node problem recently introduced in [34]. An application of this problem is, e.g., to stop the spread of wildfire in a forest as well as to limit the effect of malicious viral attacks on a network. In both cases, given a defender policy, the resulting optimization problem is a bilevel program in which the leader and the follower operate on a given undirected graph $G = (V, E)$: the leader represents the malicious player who wants to infect the maximum number of nodes in the network, while the follower tries to minimize this figure. Both the leader and the follower can select nodes according to some given budget. In particular, the leader can infect some nodes, from which infection propagates through the edges of the graph, until it reaches a node that has been protected by the follower. Random instances with number of nodes $|V| \in \{25, 50, 80\}$ and budget for the leader b up to 5 were generated in [34].

6.2.2 Generalized Interdiction Instances

In this section we describe the procedure we used to randomly generate GIP instances. Observe that, when no specific structure of the bilevel problem is imposed, one can easily end up with an infeasible or unbounded instance. Additionally, one may encounter instances for which the optimal solution is unattainable; see, e.g., [38, 39] for a discussion of this behavior. A main issue is that, for any given feasible $\bar{x}$ of the leader, the follower may be infeasible, resulting in an infeasible problem. Additionally, it may happen that for a given $\bar{x}$, an associated optimal follower solution, say $\bar{y}$, is such that $(\bar{x}, \bar{y})$ is not feasible due to the leader constraints (11).

To avoid the pathological cases above, we generated a class GEN of random GIP problems using the following approach. Instances are constructed by specifying five parameters: number of leader variables $|N_x|$, number of follower variables $|N_y|$, number of interdiction variables $|N|$, number of leader constraints $|C_L|$ (all in $\leq$ form), and number of follower constraints $|C_F|$ (all in $\leq$ form). All variables are binary, and coefficients for the objectives and the constraints are taken uniformly random as integers in $[-50, 50]$. The right-hand side of each constraint is taken as $\lfloor \frac{\alpha}{100} \cdot \Sigma \rfloor$, where α is an integer taken uniformly random from $[25, 75]$, and Σ is either the sum of all positive or negative coefficients of the currently considered row (both with 50% probability). Each leader constraint has an additional binary slack x-variable, with coefficient $-100,000$ in the constraint and penalty $100,000$ in the leader objective. Similarly, each follower constraint has a continuous slack y-variable with coefficient -1 in the constraint, and penalty $-100,000$ in the follower objective. We disregarded all instances for which the HPR did not admit a feasible solution without leader slack-variables. We created ten instances for each allowed combination in $|N_x| \in \{20, 30\}$, $|N_y| \in \{20, 30\}$, $|N| \in \{10, 20, 30\}$, and $|C_L| = |C_F| = 20$. This resulted in 90 instances (note that the interdiction variables need to be a subset of the leader/follower variables).

6.3 Results on instances from literature

We first evaluated the performance of our proposed heuristics on the instances from the literature. For all those instances, the optimal solution value was computed in [4], thus allowing us to evaluate the quality of the heuristic solutions. Note that, in some cases, the exact method proposed in [4] required one or more hours of computing time to find a provably optimal solution.

We ran each heuristic algorithm with a very short time limit of just 10 seconds. Table 2 gives the number of instances for which each algorithm returned an optimal solution (without proving its optimality of course), as well as the average percentage gap between the heuristic and the optimal solution values. For each instance, the optimality gap is computed as $100 \cdot |z^{heu} - z^{opt}|/(|z^{opt}| + 10^{-10})$, where z^{heu} and z^{opt} denote the heuristic and optimal solution values, respectively.

Table 2: Results of the heuristics on the instances from the literature, with a time limit of 10 sec.s. Column #opt gives the number of times a heuristic found the optimal value z^{opt}, and %gap gives the average primal gap, computed as $100 \cdot |z^{heu} - z^{opt}|/(|z^{opt}| + 10^{-10})$. $^{(*)}$Global statistics do not include class TRSC+.

		GD		OS		OS+		GR		I		I+		DR+	
set	#inst	#opt	%gap	#opt	%gap	#opt	%gap	#opt	%gap	#opt	%gap	#opt	%gap	#opt	%gap
CCLW	50	5	17.56	44	0.12	49	0.00	10	6.93	50	0.00	50	0.00	50	0.00
TRSK	150	37	11.79	93	2.00	140	0.35	79	3.82	150	0.00	150	0.00	149	0.08
D	160	58	14.24	154	0.16	160	0.00	115	1.89	160	0.00	160	0.00	160	0.00
SAC	144	7	19.68	121	0.14	136	0.07	30	8.75	142	0.04	141	0.05	143	0.00
TRSC	80	1	98.96	5	73.96	14	44.17	5	70.62	22	37.92	26	29.79	52	14.17
TRSC+	80	1	98.96	23	36.25	30	26.87	6	69.58	42	20.62	50	14.58	53	12.08
FIRE	72	5	30.43	15	29.69	52	5.68	20	13.27	53	5.67	56	4.24	55	4.58
sum$^{(*)}$	656	113		432		551		259		577		583		609	
average$^{(*)}$			32.11		17.68		8.38		17.55		7.27		5.68		3.14

The leftmost part of Table 2 refers to the three algorithms (GD, OS, and OS+) that can terminate before the imposed time limit; as a matter of fact, they were very fast in this test and often required one second or less. The results in the table show that algorithm OS performs much better than GD: while OS is able to compute an optimal solution in 65% of the instances and has an average percentage gap of about 17%, GD optimally solves only 17% of the instances and has an average gap of about 32%. As expected, OS+ has an even (much) better performance, in that it is able to compute an optimal solution in 83% of the instances and has an average gap of about 8%. The rightmost part of the table considers more sophisticated heuristics that run until the time limit is hit. The randomized version of the greedy (algorithm GR) is clearly the worst method in this class (only 39% optimal solutions found with an average gap of about 17%), while the other methods are able to produce an optimal solution for more than 87% of the instances, with an average gap below 7%. Algorithm DR+ is the clear winner (92% of the instances solved to optimality, with an average error of about 3%), though very good results can be obtained also by using algorithm I+ (88% of the instances solved to optimality, average error of about 5%).

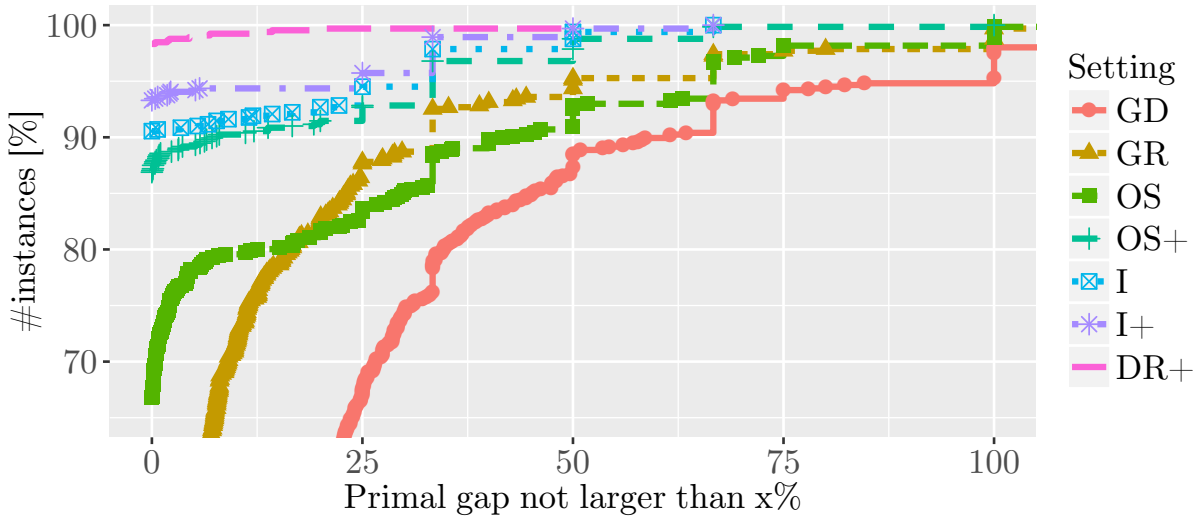
Observe that the global results in Table 2 (lines "sum" and "average") do not include instances in class TRSC+, though the table reports the results for both classes TRSC and TRSC+ to stress the importance of having a follower formulation with a small integrality gap. As expected, all our heuristics perform significantly better when the improved formulation is considered, with the only exception of DR+ whose performance is already quite good even without the additional inequalities (53)—recall that DR+ is able to automatically improve the follower formulation by generating valid cuts on the fly.

It is worth noting that all our heuristics have an extremely good performance on the first four problem classes, and a very good one on the last class. For the clique-based instances TRSC, instead, even our best method (DR+) has an average error of more than 10%. This is partly explained by the fact that the optimal solution value for these instances typically is a very small integer (less than 5 in most cases), so an error of just one unit translates into a very large percentage error of 20% or more. A similar consideration applies (to a lesser extent) to FIRE as well, where the optimal values are small integers in range 2 to 63.

Figure 1 plots the performance profile [40] of the percentage gap of the seven heuristics on the same

instances of Table 2, and confirms the relative ranking among the compared heuristics.

Figure 1: Performance profile plot of the percentage gap w.r.t. the optimal solution z^{opt}, computed as $100 \cdot |z^{heu} - z^{opt}|/(|z^{opt}| + 10^{-10})$, on the instances from the literature, with a time limit of 10 sec.s.



6.4 Results on new instances

In this section we address the newly proposed SIP instances, namely those in sets `LKIP` and `LCIP`, as well as the GIP instances (set `GEN`). As for these instances the optimal solution value is not known, we evaluated solution quality by comparison with the value, say z^{best}, of the best solution found by the heuristics under comparison. For these instances, all algorithms were executed with a time limit of 60 seconds.

Table 3 reports the outcome of the new experiments. In particular, column "%gap" reports the average percentage gap with respect to z^{best}, while column "#bst"gives the number of instances for which the algorithm produced the best solution. The upper part of the table gives results for SIP instances, whereas the last line refers to the GIP instances of class `GEN`. As the benchmark algorithms `GD` and `GR` are not designed for GIP instances, we do not report the associated results in this case. In addition, we do not report the percentage gap for these instances, as this figure can be extremely large due to the very large objective coefficient of the slack variables.

Table 3: Results of the heuristics on the new instances, with a time limit of 60 sec.s. Column #bst gives the number of times a heuristic found the best value z^{best}, and %gap gives the average primal gap, computed as $100 \cdot |z^{heu} - z^{best}|/(|z^{best}| + 10^{-10})$ for each instance. For instance set `GEN` the average primal gap is not reported, as it may be very large due to slack variable coefficients in the objective function.

		GD		OS		OS+		GR		I		I+		DR+	
set	#inst	#bst	%gap	#bst	%gap	#bst	%gap	#bst	%gap	#bst	%gap	#bst	%gap	#bst	%gap
LKIP	400	59	20.31	393	0.00	400	0.00	60	19.14	400	0.00	400	0.00	400	0.00
LCIP	60	0	63.93	0	53.75	34	17.91	0	57.21	1	34.14	40	7.05	58	0.32
sum	460	59		393		434		60		401		440		458	
average			42.12		26.87		8.96		38.17		17.07		3.53		0.16
GEN	90	-	-	24	-	44	-	-	-	49	-	49	-	90	-

Figures 2 and 3 plot the performance profile of the percentage gaps computed with respect to z^{best} for newly proposed SIP and GIP instances, respectively.

Figure 2: Performance profile plot of the percentage gap w.r.t. the best heuristic solution z^{best}, computed as $100 \cdot |z^{heu} - z^{best}|/(|z^{best}| + 10^{-10})$, on the instances LCIP and LKIP with a time limit of 60 sec.s.

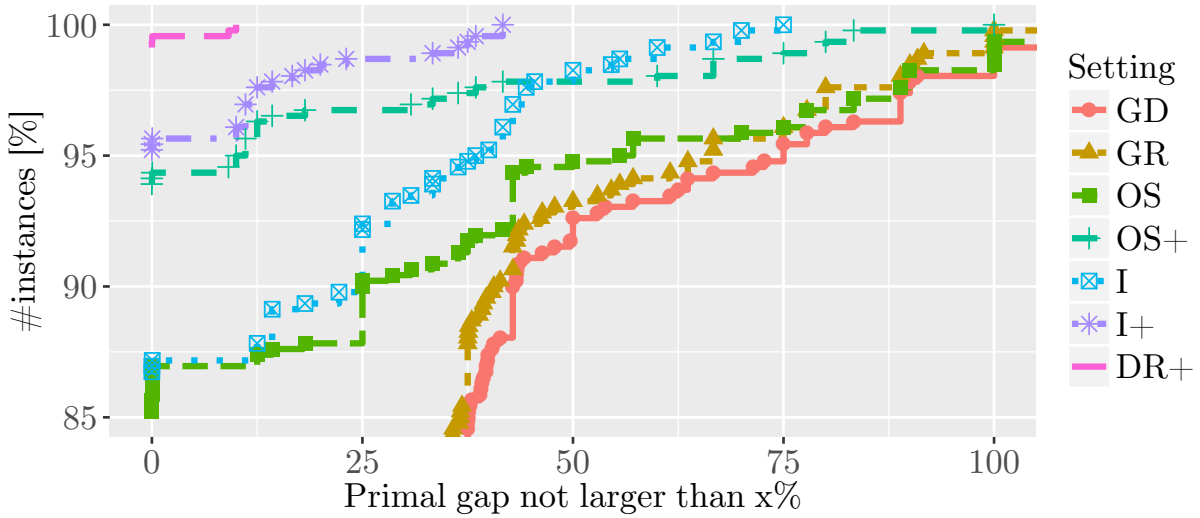
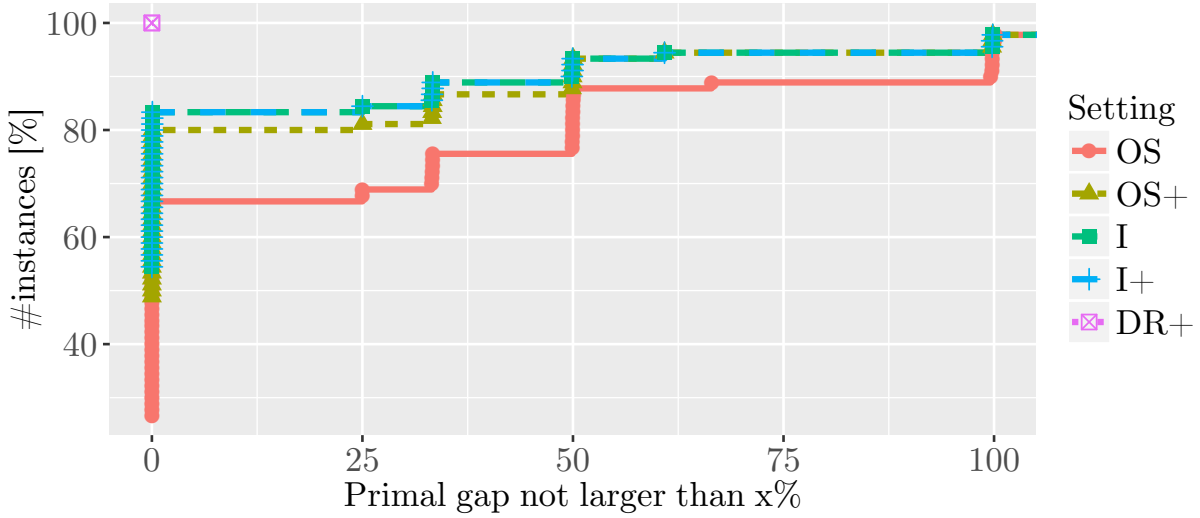


Figure 3: Performance profile plot of the percentage gap w.r.t. the best heuristic solution z^{best}, computed as $100 \cdot |z^{heu} - z^{best}|/(|z^{best}| + 10^{-10})$, on the instance set GEN with a time limit of 60 sec.s. Our DR+ heuristic found the best solution in 100% of the instences.



The results show that all our heuristics perform equally well on LKIP, while for LCIP and GEN our more advanced version (DR+) has much better performance than all the competing methods. For all instances in this benchmark too, the greedy algorithm is clearly outperformed by the other heuristics, including the basic version described in Section 5.1. Finally, observe that algorithm DR+ is consistently the best approach for GEN instances.

7 Conclusions and directions of future work

In this paper we have considered Generalized Interdiction Problems (GIPs), a special case of Mixed-Integer Bilevel Linear Programs (MIBLP) that has important practical applications. Generalized interdiction can be seen as Stackelberg game where two players (called the leader and the follower) share a set of items, and the leader can interdict the usage of certain items by the follower. These problems can be modeled as Mixed-Integer Nonlinear Programming problems, whose exact solution can be very challenging.

We have proposed a single-level (compact) Mixed-Integer Linear Programming (MILP) reformulation of the problem obtained from GIP by relaxing the integrality of the follower variables, to be used within a heuristic solution approach. Three different heuristics of increasing sophistication level have been described and computationally analyzed on a large set of instances taken from the literature or generated in a random way. In particular, our new dynamic reformulation heuristic uses a mathematically-sound way to bring integrality information (in the form of valid cuts) from the follower to the global level. The outcome of our experiments is that even the simplest reformulation heuristic works well on a large number of the tested instances, often providing an optimal solution within negligible computing time. For the hardest cases, however, our new dynamic reformulation heuristic produces significantly improved results and clearly outperforms all compared methods.

Future research should address the application of our dynamic reformulation heuristic scheme to other min-max problems such as those arising in min-max regret robustness, as well as to problems where interdiction penalties are considered as, e.g., in [14]. Another interesting line of research is the application of similar heuristic schemes to general MIBLPs. Also in this context, dropping the integrality requirement on the follower variables leads to a single-level MILP reformulation, so our heuristic approach can be applied with only minor adaptations. In the general bilevel case, however, the single-level reformulation is based on KKT conditions, hence it implies hard disjunctive constraints that can make its resolution quite problematic in practice. As a matter of fact, preliminary computational results show that the bilevel heuristic performs well for some instances, but additional research effort is needed to reduce the solution time of the single-level MILP reformulation.

Acknowledgments

This research was funded by the Vienna Science and Technology Fund (WWTF) through project ICT15-014. The work of the first two authors was also supported MiUR, Italy (PRIN project), while that of the last author was also supported by the Austrian Research Fund (FWF, Project P 26755-N19).

References

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

[2] J. Outrata, On the numerical solution of a class of Stackelberg problems, ZOR - Methods and Models of Operations Research 34 (1990) 255–277.

[3] S. DeNegre, Interdiction and discrete bilevel linear programming, Ph.D. thesis, Lehigh University (2011).

[4] M. Fischetti, I. Ljubić, M. Monaci, M. Sinnl, An improved branch-and-cut algorithm for mixed-integer bilevel linear programs, Technical Report, DEI, University of Padova (2016).

[5] H. Von Stackelberg, The theory of the market economy, Oxford University Press, 1952.

[6] G. Brown, M. Carlyle, J. Salmeron, K. Wood, Analyzing the vulnerability of critical infrastructure to attack and planning defenses, Tutorials in Operations Research: Emerging Theory, Methods, and Applications (2005) 102–123.

[7] G. Brown, M. Carlyle, J. Salmeron, K. Wood, Defending critical infrastructure, Interfaces 36 (6) (2006) 530–544.

[8] A. Washburn, K. Wood, Two-person zero-sum games for network interdiction, Operations Research 43 (2) (1995) 243–251.

[9] G. Brown, M. Carlyle, R. Harney, E. Skroch, K. Wood, Interdicting a nuclear-weapons project, Operations Research 57 (4) (2009) 866–877.

[10] D. P. Morton, F. Pan, K. J. Saeger, Models for nuclear smuggling interdiction, IIE Transactions 39 (1) (2007) 3–14.

[11] A. Lodi, T. Ralphs, F. Rossi, S. Smriglio, Interdiction branching, Technical Report, COR@L Laboratory, Lehigh University (2011).

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

[13] A. Caprara, M. Carvalho, A. Lodi, G. Woeginger, Bilevel knapsack with interdiction constraints, INFORMS Journal on Computing 28 (2) (2016) 319–333.

[14] E. Israeli, R. Wood, Shortest-path network interdiction, Networks 40 (2) (2002) 97–111.

[15] C. Lim, J. C. Smith, Algorithms for discrete and continuous multicommodity flow network interdiction problems, IIE Transactions 39 (1) (2007) 15–26.

[16] R. K. Wood, Deterministic network interdiction, Mathematical and Computer Modelling 17 (2) (1993) 1–18.

[17] F. Furini, M. Iori, S. Martello, M. Yagiura, Heuristic and exact algorithms for the interval min–max regret knapsack problem, INFORMS Journal on Computing 27 (2) (2015) 392–405.

[18] L. Assunção, T. F. Noronha, A. C. Santos, R. Andrade, A linear programming based heuristic framework for min-max regret combinatorial optimization problems with interval costs, Computers & Operations Research 81 (2017) 51–66.

[19] C. Bazgan, S. Toubaline, D. Vanderpooten, Efficient determination of the k most vital edges for the minimum spanning tree problem, Computers & Operations Research 39 (11) (2012) 2888–2898.

[20] D. R. Fulkerson, G. C. Harding, Maximizing the minimum source-sink path subject to a budget constraint, Mathematical Programming 13 (1) (1977) 116–118.

[21] Y. Tang, J.-P. Richard, J. Smith, A class of algorithms for mixed-integer bilevel min–max optimization, Journal of Global Optimization (2015) 1–38.

[22] R. Wood, Bilevel Network Interdiction Models: Formulations and Solutions, John Wiley & Sons, Inc., 2010.

[23] C. Bazgan, S. Toubaline, Z. Tuza, The most vital nodes with respect to independent set and vertex cover, Discrete Applied Mathematics 159 (17) (2011) 1933 – 1946.

[24] M. Fischetti, I. Ljubić, M. Monaci, M. Sinnl, Interdiction games and monotonicity, Technical Report, DEI, University of Padova (2016).

[25] H. Held, D. L. Woodruff, Heuristics for multi-stage interdiction of stochastic networks, Journal of Heuristics 11 483–500.

[26] M. Gendreau, P. Marcotte, G. Savard, A hybrid tabu-ascent algorithm for the linear bilevel programming problem, Journal of Global Optimization 8 (3) 217–233.

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

[28] U. P. Wen, A. D. Huang, A simple tabu search method to solve the mixed-integer linear bilevel programming problem, European Journal of Operational Research 88 (3) 563–571.

[29] H. I. Calvete, C. Gal, P. M. Mateo, A new approach for solving linear bilevel problems using genetic algorithms, European Journal of Operational Research 188 (1) 14–28.

[30] E.-G. Talbi, Metaheuristics for bi-level optimization, Vol. 482, Springer, 2013.

[31] M. Fischetti, I. Ljubić, M. Monaci, M. Sinnl, Intersection cuts for bilevel optimization, in: Q. Louveaux, M. Skutella (Eds.), Integer Programming and Combinatorial Optimization: 18th International Conference, IPCO 2016, Liège, Belgium, June 1-3, 2016, Proceedings, Springer International Publishing, Extended version submitted for publication, 2016, pp. 77–88.
URL http://www.dei.unipd.it/~fisch/papers/intersection_cuts_for_bilevel_optimization.pdf

[32] E. Rothberg, An evolutionary algorithm for polishing mixed integer programming solutions, INFORMS Journal on Computing 19 (4) (2007) 534–541.

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

[34] A. Baggio, M. Carvalho, A. Lodi, A. Tramontani, Multilevel approaches for the critical node problem, working paper, École Polytechnique de Montréal (2016).

[35] S. Martello, D. Pisinger, P. Toth, Dynamic programming and strong bounds for the 0-1 knapsack problem, Management Science 45 (3) (1999) 414–424.

[36] S. Khuri, T. Baeck, J. Heitkoetter, SAC94 Suite: Collection of Multiple Knapsack Problems, http://www.cs.cmu.edu/Groups/AI/areas/genetic/ga/test/sac/0.html (1994).

[37] A. Caprara, M. Fischetti, {0, 1/2}-Chvátal-Gomory cuts, Mathematical Programming 74 (3) (1996) 221–235.

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

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

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