Mathematical Programming Computation (2020) 12:529–568
https://doi.org/10.1007/s12532-020-00183-6

FULL LENGTH PAPER

A branch-and-cut algorithm for mixed integer bilevel linear optimization problems and its implementation

Sahar Tahernejad[1] · **Ted K. Ralphs**[1] · **Scott T. DeNegre**[2]

Received: 25 April 2017 / Accepted: 14 November 2019 / Published online: 26 June 2020

Abstract
In this paper, we describe a comprehensive algorithmic framework for solving mixed integer bilevel linear optimization problems (MIBLPs) using a generalized branch-and-cut approach. The framework presented merges features from existing algorithms (for both traditional mixed integer linear optimization and MIBLPs) with new techniques to produce a flexible and robust framework capable of solving a wide range of bilevel optimization problems. The framework has been fully implemented in the open-source solver `MibS`. The paper describes the algorithmic options offered by `MibS` and presents computational results evaluating the effectiveness of the various options for the solution of a number of classes of bilevel optimization problems from the literature.

Keywords Bilevel optimization · Mixed integer optimization · Branch-and-cut algorithm · Open-source solver

1 Introduction

This paper describes an algorithmic framework for the solution of *mixed integer bilevel linear optimization problems* (MIBLPs) and `MibS`, its open-source software implementation. MIBLPs comprise a difficult class of optimization problems that arise in

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s12532-020-00183-6) contains supplementary material, which is available to authorized users.

✉ Ted K. Ralphs
ted@lehigh.edu

Sahar Tahernejad
sat214@lehigh.edu

Scott T. DeNegre
DeNegreS@hss.edu

[1] Department of Industrial and System Engineering, Lehigh University, Bethlehem, PA, USA

[2] Hospital for Special Surgery, New York, NY, USA

applications in which multiple, possibly competing decision-makers (DMs), make a sequence of decisions over time. For an ever-increasing array of such applications, the traditional framework of mathematical optimization, which assumes a single DM with a single objective function making a decision at a single point in time, is inadequate.

The motivation for the development of `MibS`, which was begun a decade ago, is both to serve as an open test bed for new algorithmic ideas and to provide a platform for solution of the wide variety of practical problems of this type currently coming under scientific study. The modeling framework underlying `MibS` is that of *multilevel optimization*. Multilevel optimization problems model applications in which decisions are made in a sequence, with decisions made earlier in the sequence affecting the options available later in the sequence. Under the assumption that all DMs are rational and have complete information about their own and each other's models and input data (there is no stochasticity), we can describe such problems formally using the language of mathematical optimization. To do so, we consider a set of decision variables, partitioned into subsets associated with individual DMs. Conceptually, these sets of decision variables are ordered according to the sequence in which decisions are to be made. We use the term *level* (or *stage*, in some contexts) to denote each DM's position in the sequence. The term is intended to conjure up a hierarchy in which decisions flow from top to bottom, so decisions earlier in the sequence are said to be "at a higher level." The decisions made by higher-level DMs influence those of lower-level DMs through a parametric dependence of the lower-level decision problems on higher-level decisions. Thus, higher-level decisions must be made taking into account the effect those decisions will have on lower-level decisions, which in turn impact the objective value of the higher-level solution.

The decision hierarchy of a national government provides an archetypal example. The national government makes decisions about tax rates or subsidies that in turn affect the decisions of state and local governments, which finally affect the decisions of individual taxpayers. Since later decisions affect the degree to which the national government achieves its original objective, the decisions at that higher level must be made in light of the reactions of lower-level DMs to those decisions [1].

Thanks to the steady improvement in solution methodologies for traditional optimization problems and the availability of open-source solvers [2], the development of practical solution methods for MIBLPs has become more realistic. The availability of practical solution methods has in turn driven an accompanying increase in demand for such methodologies. The literature is now replete with applications requiring the solution of such models [3–6]. In the remainder of this section, we introduce the basic framework of bilevel optimization.

1.1 Mixed integer bilevel optimization

In this section, we formally introduce MIBLPs, the class of multilevel optimization problem in which we have only two levels and two associated DMs. The decision variables of an MIBLP are partitioned into two subsets, each conceptually controlled by one of the DMs. We generally denote by x the variables controlled by the *first-level DM* or *leader* and require that the values of these variables be contained in the set

$X = \mathbb{Z}_+^{r_1} \times \mathbb{R}_+^{n_1 - r_1}$ representing integrality constraints. We denote by y the variables controlled by the *second-level DM* or *follower* and require the values of these variables to be in the set $Y = \mathbb{Z}_+^{r_2} \times \mathbb{R}_+^{n_2 - r_2}$. Throughout the paper, we refer to a sub-vector of $x \in \mathbb{R}^{n_1}$ indexed by a set $K \subseteq \{1, \ldots, n_1\}$ as x_K (and similarly for sub-vectors of $y \in \mathbb{R}^{n_2}$).

The general form of an MIBLP is

$$\min_{x \in X} \{cx + \Xi(x)\}, \tag{MIBLP}$$

where the function Ξ is a *risk function* that encodes the part of the objective value of x that depends on the response to x in the second level. This function may have different forms, depending on the precise variant of the bilevel problem being solved. In this paper, we focus on the so-called *optimistic* case [7], in which

$$\Xi(x) = \min \left\{ d^1 y \;\middle|\; y \in \mathcal{P}_1(x),\, y \in \operatorname{argmin} \left\{ d^2 \hat{y} \;\middle|\; \hat{y} \in \mathcal{P}_2(x) \cap Y \right\} \right\}, \tag{RF-OPT}$$

where

$$\mathcal{P}_1(x) = \left\{ y \in \mathbb{R}_+^{n_2} \;\middle|\; G^1 y \geq b^1 - A^1 x \right\}$$

is a parametric family of polyhedra containing points satisfying the linear constraints of the first-level problem with respect to a given $x \in \mathbb{R}^{n_1}$ and

$$\mathcal{P}_2(x) = \left\{ y \in \mathbb{R}_+^{n_2} \;\middle|\; G^2 y \geq A^2 x \right\}$$

is a second parametric family of polyhedra containing points satisfying the linear constraints of the second-level problem with respect to a given $x \in \mathbb{R}^{n_1}$. The input data is $A^1 \in \mathbb{Q}^{m_1 \times n_1}$, $G^1 \in \mathbb{Q}^{m_1 \times n_2}$, $b^1 \in \mathbb{Q}^{m_1}$, $A^2 \in \mathbb{Q}^{m_2 \times n_1}$ and $G^2 \in \mathbb{Q}^{m_2 \times n_2}$. As is customary, we define $\Xi(x) = \infty$ in the case of either $x \notin X$ or infeasibility of the problem on the right-hand side of (RF-OPT).

Note that it is typical in the literature on bilevel optimization for the parametric right-hand side of the second-level problem to include a fixed component, i.e., to be of the form $b^2 - A^2 x$ for some $b^2 \in \mathbb{Q}^{m^2}$. The form introduced here is more general, since adding a constraint $x_1 = 1$ to the upper level problem results in a problem equivalent to the usual one. The advantage of the form here is that it results in a risk function that is subadditive in certain important cases (though not in general), a desirable property for reasons that are beyond the scope of this paper.

As we mentioned above, other variants of the risk function are possible and the algorithm we present can be adapted to these cases (see Sect. 3.3). A pessimistic risk function can be obtained simply by considering

$$\Xi(x) = \max \left\{ d^1 y \;\middle|\; y \in \mathcal{P}_1(x),\, y \in \operatorname{argmin} \left\{ d^2 \hat{y} \;\middle|\; \hat{y} \in \mathcal{P}_2(x) \cap Y \right\} \right\}. \tag{RF-PES}$$

Note that according to this definition, the second-level solution y is required to be a member of $\mathcal{P}_1(x)$ (i.e., be feasible for the upper-level constraints), though there could exist $y \in \operatorname{argmin}\{d^2y \mid y \in \mathcal{P}_2(x) \cap Y\} \setminus \mathcal{P}_1(x)$. Allowing such second-level solutions to be chosen is also possible and can be accommodated within the framework of (RF-PES) by the addition of dummy variables at the first level. The details are beyond the scope of this paper.

It should be pointed out that we explicitly allow the second-level variables to be present in the first-level constraints. This allowance is rather non-intuitive and leads to many subtle algorithmic complications. Nevertheless, there are applications in which it is necessary to allow this and since `MibS` does allow this possibility, we felt it appropriate to express the results in this full generality. The reader should keep in mind, however, that many of the ideas discussed herein can be simplified in the more usual case that $G^1 = 0$ and we endeavor to point this out in particular cases.

Note that the formulation (MIBLP) does not explicitly involve the second-level variables. The reason for expressing the formulation in this way is to emphasize two things. First, it emphasizes that the goal of solving (MIBLP) is to determine the optimal values of the first-level variables only. Thus, we would ideally like to "project out" the second-level variables. In contrast to the Benders method for traditional optimization problems, however, the second-level variables are re-introduced in solving the relaxation that we employ in our branch-and-cut algorithm (see Sect. 2.1). Second, it is necessary at certain points in the algorithm to evaluate $\Xi(x)$, and it is convenient to give it an explicit form here to make this step clearer.

MIBLPs also have an alternative formulation that employs the *value function* of the second-level problem and explicitly includes the second-level variables. This formulation is given by

$$\min\left\{cx + d^1y \;\middle|\; x \in X, y \in \mathcal{P}_1(x) \cap \mathcal{P}_2(x) \cap Y, d^2y \leq \phi(A^2x)\right\}, \quad \text{(MIBLP-VF)}$$

where ϕ is the so-called *value function* of the second-level problem, which is a standard mixed integer linear optimization problem (MILP). The value function returns the optimal value of second-level problem for a given right-hand side, defined by

$$\phi(\beta) = \min\left\{d^2y \;\middle|\; G^2y \geq \beta, y \in Y\right\} \quad \forall \beta \in \mathbb{R}^{m_2}. \quad \text{(VF)}$$

From this alternative formulation, it is evident that if the values of the first-level variables are fixed in (MIBLP-VF) (i.e., the constraints imply that their values must be constant), then the problem of minimizing over the remaining variables is an MILP. In fact, it is not difficult to observe that only the first-level variables having non-zero coefficients in the second-level constraints need be fixed in order for $\phi(A^2x)$ to be rendered a constant and for (MIBLP-VF) to reduce to an MILP. This result is stated formally in Sect. 2.2. Because of their central importance in what follows, we formally define the concept of *linking variables*.

Definition 1 (*Linking variables*) Let

$$L = \left\{i \in \{1, \ldots, n_1\} \;\middle|\; A_i^2 \neq 0\right\},$$

be the set of indices of first-level variables with non-zero coefficients in the second-level problem, where A^2_i denotes the ith column of matrix A^2. We refer to such variables as *linking variables*.

Per our earlier notation, x_L is the sub-vector of $x \in \mathbb{R}^{n_1}$ corresponding to the linking variables. By assuming all linking variables are integer variables, we assure the existence of an optimal solution [8].

Assumption 1 $L = \{1, \ldots, k_1\}$ for $k_1 \leq r_1$.

We note here that it can in fact be assumed without loss of generality that $k_1 = r_1$ by simply moving the non-linking variables to the second level. While this is conceptually inconsistent with the intent of the original model, it is not difficult to see that the resulting model is *mathematically* equivalent.
In what follows, it will be convenient to refer to the set

$$\mathcal{P} = \left\{(x, y) \in \mathbb{R}^{n_1 \times n_2}_{+} \;\middle|\; y \in \mathcal{P}_1(x) \cap \mathcal{P}_2(x)\right\}$$

of all points satisfying the non-negativity and linear inequality constraints at both levels and the set

$$\mathcal{S} = \mathcal{P} \cap (X \times Y)$$

of points in $\mathcal{P}$ that also satisfy integrality restrictions.

Assumption 2 $\mathcal{P}$ is bounded.

This assumption is made to simplify the exposition, but is easy to relax in practice. Corresponding to each $x \in \mathbb{R}^{n_1}$ with $x_L \in \mathbb{Z}^L$, we have the *rational reaction set*, which is defined by

$$\mathcal{R}(x) = \operatorname{argmin} \left\{d^2 y \;\middle|\; y \in \mathcal{P}_2(x) \cap Y\right\}.$$

This set may be empty either because $\mathcal{P}_2(x) \cap Y$ is itself empty or because there exists $r \in \mathbb{R}^{n_2}_{+}$ such that $G^2 r \geq 0$ and $d^2 r < 0$ (in which case the second-level problem is unbounded no matter what first-level solution is chosen). The latter case can be easily detected in a pre-processing step, so we assume w.l.o.g. that this does not occur (note that this case cannot occur when $G^1 = 0$, since Assumption 2 would then imply that $\left\{r \in \mathbb{R}^{n_2}_{+} \setminus 0 \;\middle|\; G^2 r \geq 0\right\} = \emptyset$).

Assumption 3 $\left\{r \in \mathbb{R}^{n_2}_{+} \;\middle|\; G^2 r \geq 0, d^2 r < 0\right\} = \emptyset$.

Under our assumptions, the *bilevel feasible region* (with respect to the first- and second-level variables in (MIBLP-VF)) is

$$\mathcal{F} = \{(x, y) \in X \times Y \mid y \in \mathcal{P}_1(x) \cap \mathcal{R}(x)\}$$

and members of $\mathcal{F}$ are called *bilevel feasible solutions*. Although the ostensible goal of solving (MIBLP) is to determine $x \in X$ minimizing $cx + \Xi(x)$, this is equivalent to finding a member of $\mathcal{F}$ that optimizes the first-level objective function $cx + d^1 y$. Observe, however, that by this definition of feasibility, we may have $x^* \in X$ that is optimal for (MIBLP), while for some $\hat{y} \in Y$, $(x^*, \hat{y}) \in \mathcal{F}$ but $\Xi(x^*) < d^1 \hat{y}$.

Because we consider the problem to be that of determining the optimal first-level solution, it is also useful to denote the feasible set with respect to first-level variables only as

$$\mathcal{F}_1 = \text{proj}_x(\mathcal{F}).$$

For $x \in X$, we have that

$$x \in \mathcal{F}_1 \Leftrightarrow x \in \text{proj}_x(\mathcal{F}) \Leftrightarrow \mathcal{R}(x) \cap \mathcal{P}_1(x) \neq \emptyset \Leftrightarrow \Xi(x) < \infty$$

and we say that $x \in \mathbb{R}^{n_1}$ is feasible if $x \in \mathcal{F}_1$. We can then interpret the conditions for bilevel feasibility of (x, y) as consisting of the following two properties of the first- and second-level parts of the solution independently.

Feasibility Condition 1 $x \in \mathcal{F}_1$.

Feasibility Condition 2 $y \in \mathcal{P}_1(x) \cap \mathcal{R}(x)$.

We exploit this notion of feasibility later in our algorithm.

A related set (see Fig. 1) is the set of feasible solutions to the *bilevel linear optimization problem* (BLP) that results from discarding the integrality restrictions, defined as

$$\mathcal{F}_{\text{LP}} = \left\{ (x, y) \in \mathbb{R}_+^{n_1 \times n_2} \;\middle|\; y \in \mathcal{P}_1(x) \cap \mathcal{R}_{\text{LP}}(x) \right\},$$

where

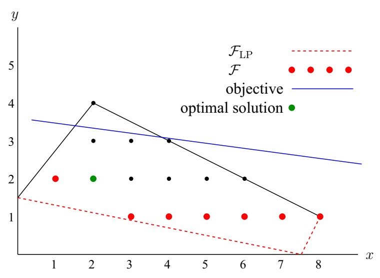
$$\mathcal{R}_{\text{LP}}(x) = \text{argmin} \left\{ d^2 y \;\middle|\; y \in \mathcal{P}_2(x) \right\}.$$

Note that we do *not* have in general that $\mathcal{F} \subseteq \mathcal{F}_{\text{LP}}$, as discussed later in Sect. 1.3.

1.2 Special cases

There are a number of cases for which `MibS` has specialized methods. One of the most important special cases is the *zero sum* case in which $d^1 = -d^2$. The *mixed integer interdiction problem* (MIPINT) is a specific subclass of zero sum problem in which the first-level variables are binary and are in one-to-one correspondence with the second-level variables ($n = n_1 = n_2$). When a first-level variable is fixed to one, this prevents the associated variable in the second-level problem from taking a non-zero value. The interdiction problem can be formulated as [9,10]

$$\min \left\{ d^1 y \;\middle|\; x \in \mathcal{P}_1^{INT} \cap \mathbb{B}^n,\, y \in \text{argmax} \left\{ d^1 y \;\middle|\; y \in \mathcal{P}_2^{INT}(x) \cap Y \right\} \right\}, \quad \text{(MIPINT)}$$



$$\begin{aligned} \min_{x \in \mathbb{Z}_+} \quad & -x - 10y \\ \text{s.t.} \quad & y \in \operatorname{argmin} \{ y : \\ & \qquad -25x + 20y \leq 30 \\ & \qquad x + 2y \leq 10 \\ & \qquad 2x - y \leq 15 \\ & \qquad 2x + 10y \geq 15 \\ & \qquad y \in \mathbb{Z}_+ \ \} \end{aligned}$$

Fig. 1 The feasible region of IBLP [12]

where

$$\mathcal{P}_1^{INT} = \left\{ x \in \mathbb{R}_+^n \;\middle|\; A^1 x \geq b^1 \right\},$$

$$\mathcal{P}_2^{INT}(x) = \left\{ y \in \mathbb{R}_+^n \;\middle|\; G^2 y \geq b^2, y \leq \text{diag}(u)(e - x) \right\},$$

e represents an n-dimensional vector of ones and $u_i \in \mathbb{R}$ denotes the upper bound of y_i for $i = 1, \ldots, n$. The special structure of MIPINTs can be employed to develop specialized methods for these problems.

1.3 Computational challenges

From the point of view of both theory and practice, bilevel problems are difficult to solve. From the perspective of complexity theory, the problem is NP-hard even for the case in which all variables are continuous ($r_1 = r_2 = 0$). The general case is in the class Σ_2^p-hard [11], which is the class of problems that can be solved in non-deterministic polynomial time, given an NP oracle. To put it in terms that are a little less formal, these are problems for which even the problem of checking feasibility of a given point in $X \times Y$ is complete for NP in general. Alternatively, simply computing $\Xi(x)$ for $x \in X$ (determining its objective function value), is also NP-hard.

Because determining whether a given solution is feasible is an NP-complete problem, solution of MIBLPs is inherently more difficult than solution of the more familiar case of MILP (equivalent to the case of $L = \emptyset$). Search algorithms for MILPs rely on a certain amount of algorithmically guided "luck" to find high-quality solutions quickly. This works reasonably well in this simpler case because checking feasibility of any given solution is efficient. In the case of MIBLPs, we cannot rely on this property and even if we are lucky enough to get a high-quality first-level solution, checking its feasibility is still a difficult problem. Furthermore, some of the properties whose exploitation we depend on in the case of MILP do not straightforwardly generalize to the MIBLP case. For example, removing the integrality requirement for all variables does not result in a relaxation, since relaxing the second-level problem may make solutions that were previously feasible infeasible. In fact, as we mentioned earlier, the feasible region $\mathcal{F}_{\text{LP}}$ of this BLP does not necessarily even contain the feasible region $\mathcal{F}$ of (MIBLP). Thus, even if the solution to this BLP is in $X \times Y$, it is not necessarily optimal for (MIBLP). These properties can be seen in Fig. 1, which displays a well-known example originally from Moore and Bard [12] that is well-suited for illustrating these concepts.

1.4 Previous work

Bilevel optimization has its roots in game theory and the general class of problems considered in this paper are a type of *Stackelberg game*. The Stackelberg game was studied in a seminal work by Von Stackelberg [13]. The first bilevel optimization formulations in the form presented here were introduced and the term was coined in the 1970s by Bracken and McGill [14], but computational aspects of related optimization problems have been studied since at least the 1960s (see, e.g., [15]). Study of algorithms for the case in which integer variables appear is generally acknowledged to have been launched by Moore and Bard [12], who initiated working on general MIBLPs and discussed the computational challenges of solving them. They proposed a branch-and-bound algorithm for solving MIBLPs which is guaranteed to converge if all first-level variables are integer or all second-level variables are continuous.

Until recently, most computational work focused on special cases with exploitable structure. Bard and Moore [16], Wen and Huang [17] and Faísca et al. [18] studied the bilevel problems with binary variables. Bard and Moore [16] proposed an exact algorithm for *integer bilevel optimization problems* (IBLPs) in which all variables are binary. Wen and Huang [17], on the other hand, considered MIBLPs with binary first-level variables and continuous second-level ones and suggested a *tabu search heuristic* for generating solutions to these problems. Faísca et al. [18] concentrated on MIBLPs in which all discrete variables are constrained to be binary. They reformulated the second-level problem as a multi-parametric problem with the first-level variables as parameters.

DeNegre and Ralphs [19] and DeNegre [10] generalized the existing framework of branch and cut, which is the standard approach for solving MILPs, to the case of IBLPs by introducing the *integer no-good cut* to separate bilevel infeasible solutions from the convex hull of bilevel feasible solutions (see Sect. 2.4). They also initiated development of the open-source solver `MibS`. Xu and Wang [20] focused on problems in which all first-level variables are discrete and suggested a multi-way branch-and-bound algorithm in which branching is done on the slack variables of the second-level problem. A decomposition algorithm based on column-and-constraint generation was employed by Zeng and An [21] for solving general MIBLPs. Their method finds the optimal solution if it is attainable, otherwise it finds an ϵ-optimal solution. Caramia and Mari [22] introduced a non-linear cut for IBLPs and suggested a method of linearizing this cut by the addition of auxiliary binary variables. Furthermore, they introduced a branch-and-cut algorithm for IBLPs which employs the integer no-good cut from [19]. Knapsack interdiction problems were studied by Caprara et al. [23] and they proposed an exact algorithm for these problems. Hemmati and Smith [24] formulated competitive prioritized set covering problem as an MIBLP and proposed a cutting plane algorithm for solving it. Wang and Xu [25] proposed the *watermelon algorithm* for solving IBLPs, which removes bilevel infeasible solutions that satisfy integrality constraints by a non-binary branching disjunction. Lozano and Smith [26] employed a single-level value function reformulation for solving the MIBLPs in which all first-level variables are integer. Fischetti et al. [27] suggested a branch-and-cut algorithm for MIBLPs employing a class of *intersection cuts* valid for MIBLPs under mild assumptions and developed a new family of cuts for the MIBLPs with binary first-level variables. In [28], they extended their algorithm by suggesting new types of intersection cuts and introduced the so-called *hypercube intersection cut*, valid for MIBLPs in which the linking variables are discrete. Finally, Mitsos [29] and Kleniati and Adjiman [30,31] considered the more general case of mixed integer bilevel non-linear optimization.

1.5 Overview of branch and cut

Branch and cut (a term coined by Padberg and Rinaldi [32,33]) is a variant of the well-known branch-and-bound algorithm of Land and Doig [34], the most widely-used algorithm for solving many kinds of non-convex optimization problems. For purposes of illustration, we consider here the solution of the general optimization problem

$$\min_{x \in \mathcal{X}} f(x), \tag{GO}$$

where $\mathcal{X} \subseteq \mathbb{R}^n$ is the *feasible region* and $f : \mathbb{R}^n \to \mathbb{R}$ is the *objective function*.

The approach of both the branch-and-bound and the branch-and-cut algorithms is to search the feasible region by partitioning it and then recursively solving the resulting subproblems. Implemented naively, this results in an inefficient complete enumeration. This potential inefficiency is avoided by utilizing upper and lower bounds computed for each subproblem to intelligently “prune” the search. The recursive partitioning process can be envisioned as a process of searching a rooted tree, each of whose nodes corresponds to a subproblem. The root of the tree is the node corresponding to the original problem (sometimes called the *root problem*) and the nodes adjacent to it are its *children*. The parent-child relationship applies to other nodes connected along paths in the tree in the obvious way. Nodes with no children are called *leaf nodes*. Henceforth, we denote the set of all leaf nodes in the current search tree by T.

Although it is not usually described this way, we consider the algorithm as a method for iteratively removing parts of the feasible region of the given relaxation that can be proven not to contain any *improving* feasible solutions (feasible solutions that are better than the best solution found so far) until no more such regions can be found. Although most sophisticated branch-and-bound algorithms for optimization do implicitly discard parts of the feasible region that can be shown to contain only suboptimal solutions, the algorithm we present does this more *explicitly*. The removal is done either by *branching* (partitioning the remaining set of improving solutions to define smaller subproblems) or *cutting* (adding inequalities satisfied by all improving solutions).

A crucial element of both branching and cutting is the identification of *valid disjunctions*. The notion of valid disjunction defined here, which we refer to as an *improving* valid disjunction in order to distinguish it from the more standard definition, refers to a collection of sets that contain all *improving* feasible solutions (while the standard valid disjunction should include all feasible solutions). In the remainder of the paper, we drop the word "improving" when referring to such disjunctions.

Definition 2 (*Improving valid disjunction*) An *(improving) valid disjunction* for (GO) with respect to a given $x^* \in \mathcal{X}$ is a disjoint collection

$$X_1, X_2, \ldots, X_k$$

of subsets of $\mathbb{R}^n$ such that

$$\left\{x \in \mathcal{X} \mid f(x) < f(x^*)\right\} \subseteq \bigcup_{1 \le i \le k} X_i.$$

Imposing such disjunctions is our primary method of eliminating suboptimal solutions and improving the strength of the relaxations used to produce bounds on the optimal value. They play a crucial role in defining methods of both branching and cutting, as we discuss in what follows. We now briefly describe the individual components of branch and cut that we refer to in Sect. 2.

Bounding. The most important factor needed to improve the efficiency of the algorithm is a tractable method of producing strong upper and lower bounds on the optimal solution value of each subproblem. Typically, lower bounds (assuming minimization) are obtained by solving a relaxation of the original (sub)problem. In most cases, a convex relaxation is chosen in order to ensure tractability, though with problems as difficult as MIBLPs, even a non-convex relaxation may be tractable enough to serve the desired purpose. Upper bounds are obtained by producing a solution feasible to the subproblem, either by heuristic methods or by showing that the solution of the relaxation is feasible to the original problem. We denote the lower and upper bounds associated with node t by L^t and U^t, respectively. Whenever we cannot obtain a feasible solution to a given subproblem t, we set $U^t = \infty$. Similarly, if the relaxation at node t is infeasible, we have $L^t = \infty$.

The bounds on the individual subproblems can be aggregated to obtain global bounds on the optimal solution value to the root problem. Upper bounds of all leaf nodes are aggregated to obtain the global upper bound

$$U = \min_{t \in T} U^t,$$

which represents the objective value of the best solution found so far (known as the *incumbent*). Note that this formula is a bit simplified for the purposes of presentation, since it is possible for bilevel feasible solutions that are *not* feasible for the current subproblem to be produced during the bounding step. While these do not technically update the upper bound for the subproblem, they do contribute to improvement of the global upper bound. Similarly, lower bounds for the leaf nodes are aggregated to form the global lower bound

$$L = \min_{t \in T} L^t.$$

These global upper and lower bounds are updated as the algorithm progresses and when they become equal, the algorithm terminates.

Pruning. Any node t for which $L^t \geq U$ can be discarded (*pruned*), since the feasible region of such a node cannot contain a solution with objective value lower than the current incumbent. This pruning rule implicitly subsumes two special cases. The first is when the feasible region of subproblem t is empty in which case $L^t = \infty$. The second is when $L^t = U^t \geq U$, which typically arises when solving the relaxation of the subproblem associated with node t produces a solution feasible for the original problem.

Since the global bounds are updated dynamically, this means that whether a node can be pruned or not is not a fixed property—a node that previously could not have been pruned may be pruned later when a better incumbent is found. We discuss bounding techniques in more detail in Sects. 2.1 and 2.5.

Branching. When bounds have been computed for a node and it cannot be pruned, we then partition the feasible region by identifying and imposing a valid disjunction of the form specified in Definition 2. The subproblems resulting from partitioning the feasible region $\mathcal{X}^t$ of node t are optimization problems of the form

$$\min_{x \in \mathcal{X}^t \cap X_i} f(x), \ 1 \leq i \leq k,$$

which can then be solved recursively using the same algorithm, provided that the collection $\{X_i\}_{1 \leq i \leq k}$ is chosen so as not to change the form of the optimization problem.

Typically, the disjunction is chosen so that $\bigcup_{1 \leq i \leq k} X_i$ does not contain the solution to the current relaxation, but we will see that this is not always possible in the case of MIBLP. When $\mathcal{X} \subseteq \mathbb{Z}^r \times \mathbb{R}^{n-r}$, the disjunction

$$X_1 = \left\{x \in \mathbb{R}^n \mid \pi x \leq \pi_0\right\} \quad X_2 = \left\{x \in \mathbb{R}^n \mid \pi x \geq \pi_0 + 1\right\} \tag{GD}$$

is always valid when $(\pi, \pi_0) \in \mathbb{Z}^{n+1}$ and $\pi_i = 0$ for $i > r$, since we must then have $\pi x \in \mathbb{Z}$ for all $x \in \mathcal{X}$. If the solution $\hat{x}$ to the relaxation is such that $\pi\hat{x} \notin \mathbb{Z}$, then we have $\hat{x} \notin X_1 \cup X_2$. When π is the i^{th} unit vector and $\pi_0 = \lfloor \hat{x}_i \rfloor$ (assuming $\hat{x}_i \notin \mathbb{Z}$), then the disjunction (GD) is called a *variable disjunction*. In the case of a standard MILP, at least one such disjunction must be violated whenever the solution to the relaxation is not feasible to the original problem (assuming the relaxation is a linear optimization problem (LP)). Thus, such disjunctions are all that is needed for a convergent algorithm. The situation is slightly different in the case of MIBLPs.

An important question for achieving good performance is how to choose the branching disjunctions effectively, as it is typically easy to identify many such disjunctions. A measure often used to judge effectiveness is the resulting increase in the lower bound observed after imposing the disjunction. We discuss branching strategies in more detail in Sect. 2.3.

Searching. The order in which the subproblems are considered is critically important, since, as we have already noted, the global bounds are evolving and the order in which the subproblems are considered affects their evolution. Typical options are *depth-first* (prioritize the "deepest" nodes in the tree, which tends to emphasize improvement of the upper bound by locating better incumbents) and *best-first* (prioritize those with the smallest lower bound, which tends to emphasize improvement of the lower bound). In practice, one usually employs hybrids that balance these two goals.

Cutting. The main way in which the branch-and-cut algorithm differs in its basic strategy from the branch-and-bound algorithm is that it dynamically strengthens the relaxation by adding *valid inequalities*. As with our notion of valid disjunction, we allow here for a definition slightly different than the standard one in order to allow for inequalities that remove feasible, non-improving solutions. We drop the term "improving" when discussing such inequalities in the remainder of the paper.

Definition 3 (*Improving valid inequality*) The pair $(\alpha, \beta) \in \mathbb{R}^{n+1}$ is an *(improving) valid inequality* for (GO) with respect to $x^* \in \mathcal{X}$ if

$$\left\{x \in \mathcal{X} \;\middle|\; f(x) < f(x^*)\right\} \subseteq \left\{x \in \mathbb{R}^n \;\middle|\; \alpha x \geq \beta\right\}.$$

As with branching, we typically aim to add inequalities that are violated by the solution to the current relaxation, thus removing the solution from the feasible region of the relaxation, along with some surrounding polyhedral region that contains no (improving) solutions to the original problem (or subproblem), from consideration.

Furthermore, as with branching, it is generally possible to derive a large number of valid inequalities and a choice must be made as to which ones to add to the current relaxation (adding too many can negatively impact the effectiveness of the relaxation). Also as with branching, the goal of cutting is to improve the lower bound, although selecting directly for this measure is problematic for a number of reasons. We discuss strategies for generating valid inequalities in more detail in Sect. 2.4. We refer the reader to [35] for more detailed description of the branch-and-cut algorithm.

1.6 Outline

The remainder of the paper is organized as follows. In Sect. 2, we discuss different components of the basic branch-and-cut algorithm implemented in `MibS`. Section 3 describes the overall algorithm in `MibS` and the implementation of this software. In Sect. 4, we discuss computational results. Finally, in Sect. 5, we conclude the paper with some final thoughts.

2 A branch-and-cut algorithm

The algorithm implemented in `MibS` is based on the algorithmic framework originally described by DeNegre and Ralphs [19], but with many additional enhancements. The framework utilizes a variant of the branch-and-cut algorithm and is similar in basic outline to state-of-the-art algorithms currently used for solving MILPs in practice.

The case of MIBLP is different from the MILP case in a few important ways that we discuss here at a high level before we discuss various specific components.

As we have described earlier in Sect. 1.5, one way of viewing the general approach taken by the branch-and-cut algorithm is as a method for iteratively removing parts of the feasible region of the given relaxation that can be proven not to contain any improving feasible solutions. In MILP, this is done primarily by either removing *lattice-point free* polyhedral regions (usually by imposing valid disjunctions or by generating associated valid inequalities) or by maintaining an *objective cutoff* that removes all non-improving solutions through the usual pruning process. In the MILP case, we do not generally need to explicitly track or remove solutions that are feasible to the original MILP. Such a solution can only be feasible for the relaxation at one leaf node in the search tree and is either suboptimal for the corresponding subproblem (in which case the solution must have been generated by an auxiliary heuristic) or optimal to that subproblem (in which case the node will be immediately pruned). In either case, it is unlikely the solution will arise again and there is no computational advantage to tracking it explicitly.

In MIBLP, in contrast, we may need to track and remove discrete sets of solutions. This is mainly to avoid the duplication of effort in evaluating the value function ϕ for a specific value of the linking variables. It is possible that the same computation will arise in more than one node in the search tree and re-computation should be avoided. In particular, for $x^1, x^2 \in X$, we have

$$x_L^1 = x_L^2 \Rightarrow \phi(A^2x^1) = \phi(A^2x^2).$$

Thus, tracking which sub-vectors of values for linking variables have been seen before in a pool (called the *linking solution pool*) can be computationally advantageous. We discuss the mechanism for doing this in Sect. 3.

2.1 Bounding

Lower bound. Perhaps the most important step in the branch-and-bound algorithm is that of deriving a lower bound on the optimal solution value. As we have already described, relaxing integrality restrictions does not provide an overall relaxation. However, since $\mathcal{F} \subseteq \mathcal{S} \subseteq \mathcal{P}$, relaxing the optimality condition $d^2y \leq \phi(A^2x)$ in the formulation (MIBLP-VF) results in two alternative relaxations to the original problem. The first relaxation (known as *high-point problem* [12]) is

$$\min_{(x,y)\in\mathcal{S}} cx + d^1y, \tag{R}$$

in which only the optimality condition is relaxed, while the second is

$$\min_{(x,y)\in\mathcal{P}} cx + d^1y, \tag{LR}$$

in which the integrality constraints are also relaxed. Although both are rather weak bounds, they can be strengthened by the addition of the valid inequalities described in

Sect. 2.4. Because of the enhanced tractability of (LR) and because of the advantages in warm-starting the computation during iterative computation of the bound, we use (LR) as our relaxation of choice. While it is clear that this is a relaxation of (MIBLP-VF), to see that it is a relaxation of (MIBLP), note that we have

$$d^1 y^* \leq \Xi(x^*),$$

where (x^*, y^*) is a solution to (LR).

At nodes other than the root node, we can use a similar bounding strategy. Since the branching strategy we describe shortly employs only changes to the bounds of variables, the subproblems that arise have feasible regions of the form

$$\mathcal{F}^t = \left\{ (x, y) \in \mathcal{F} \;\middle|\; l_x^t \leq x \leq u_x^t, l_y^t \leq y \leq u_y^t \right\},$$

where t is the index of the node/subproblem, and (l_x^t, u_x^t) and (l_y^t, u_y^t) represent vectors of upper and lower bounds for the first- and second-level variables, respectively. The subproblem at node t is thus the optimization problem

$$\min_{(x,y)\in\mathcal{F}^t} \quad cx + d^1 y, \tag{MIBLP^t}$$

which one can easily show is itself an MIBLP. Thus, we can apply a similar relaxation to obtain the bound

$$L^t = \min_{(x,y)\in\mathcal{P}^t} cx + d^1 y, \tag{LR^t}$$

where $\mathcal{P}^t = \{(x, y) \in \mathcal{P} \mid l_x^t \leq x \leq u_x^t, l_y^t \leq y \leq u_y^t\}$. If it can be verified that conditions for pruning are met (see Sect. 2.2), then node t should be discarded. Otherwise, the next step is to check feasibility of

$$(x^t, y^t) \in \operatorname*{argmin}_{(x,y)\in\mathcal{P}^t} \; cx + d^1 y,$$

the solution obtained when solving the relaxation.

Feasibility check. Recall that the upper bound for a given node is derived by exhibiting a feasible solution. Unlike in the case of MILP, checking whether a solution to (LR^t) is feasible for (MIBLP-VF) may itself be a difficult computational problem. Such check involves verifying that (x^t, y^t) satisfies the constraints that were relaxed: integrality conditions and second-level optimality conditions. In other words, (x^t, y^t) is bilevel feasible if and only if the following conditions are satisfied.

Feasibility Condition 3 $x^t \in X$.

Feasibility Condition 4 $y^t \in \mathcal{R}(x^t)$.

Condition 3 ensures that the integrality constraints for the first-level variables are satisfied, while Condition 4 guarantees that (x^t, y^t) satisfies the optimality constraint of second-level problem. Satisfaction of these two conditions (as opposed to the more general Conditions 1 and 2) is enough to ensure $(x^t, y^t) \in \mathcal{F}$, given that (x^t, y^t) is also a solution to (LR^t).

Verifying Condition 3 is straightforward, so this is done first. If Condition 3 is satisfied, then we consider verification of Condition 4. This involves checking both whether $y^t \in Y$ and whether $d^2 y^t = \phi(A^2 x^t)$. Checking whether $y^t \in Y$ is inexpensive so we do this first. The latter check is more expensive and is only required under conditions detailed later in Sect. 3. We may thus defer it until later, depending on the values of parameters described in Sect. 3.2.

If we decide to undertake this latter check, we first evaluate $\phi(A^2 x^t)$ to either obtain

$$\hat{y}^t \in \operatorname{argmin} \left\{ d^2 y \;\middle|\; y \in \mathcal{P}_2(x^t) \cap Y \right\}, \tag{SL-MILP}$$

or determine $\mathcal{P}_2(x^t) \cap Y = \emptyset$. The MILP (SL-MILP) is solved using an auxiliary MILP solver (more details on this in Sect. 3.4). If (SL-MILP) has a solution (as it must when $y^t \in Y$), we check whether $d^2 y^t = d^2 \hat{y}^t$. If so, (x^t, y^t) is bilevel feasible and must furthermore be an optimal solution to the subproblem at node t. In this case, $U^t = L^t$ and the current global upper bound U can be set to U^t if $U^t < U$. If (SL-MILP) has no solution, we have that $x^t \notin \mathcal{F}_1$ and x^t can be eliminated from further consideration either by branching or cutting. In fact, in this case, we can eliminate not only x^t, but any first-level solution for which $x_L = x_L^t$. We thus add x_L^t to the linking solution pool in this case.

Although it is not necessary for correctness, (SL-MILP) may be solved even when $y^t \notin Y$ (and even if $x_i^t \notin \mathbb{Z}$ for some $k_1 < i \leq r_1$), since this may still lead either to the discovery of a new bilevel feasible solution or a proof that $x^t \notin \mathcal{F}_1$. In Sect. 3, we discuss when the problem (SL-MILP) should necessarily be solved and when solving it is optional. Even in the case of infeasibility of (x^t, y^t), $(x^t, \hat{y}^t)$ is bilevel feasible (though not necessarily optimal to (MIBLP^t)) whenever $x^t \in X$ and $\hat{y}^t \in \mathcal{P}_1(x^t)$. The second condition is satisfied vacuously in the usual case of $G^1 = 0$. If $(x^t, \hat{y}^t)$ is feasible, then we can update the global upper bound if $cx^t + d^1 \hat{y}^t < U$.

Observe that solving (SL-MILP) reveals more information than the simple value of $\phi(A^2 x^t)$. Note that we have

$$\phi(A^2 x) \leq \phi(A^2 x^t) \; \forall x \in X \text{ such that } \mathcal{P}_2(x) \ni \hat{y}^t, \tag{VFB}$$

which means that $\phi(A^2 x^t)$ reveals an upper bound on the second-level problem for any other first-level solutions that admits $\hat{y}^t$ as a feasible reaction. This upper bound can be exploited in the generation of certain valid inequalities (see [10,27]) and is also the basis for an algorithm by Mitsos [29]. Furthermore, we have

$$\phi(A^2 x) = \phi(A^2 x^t) \; \forall x \in \mathbb{R}^{n_1} \text{ such that } x_L = x_L^t,$$

which means we get the *exact* value of the second-level problem with respect to certain other first-level solutions "for free". Because (i) it may improve the global upper bound,

(ii) can reveal that $x^t \notin \mathcal{F}_1$ (and, along with other solutions sharing the same values for the linking variables, should no longer be considered), and (iii) may also provide bounds on the second-level value function for other first-level solutions, (SL-MILP) may be solved in `MibS` even when (x^t, y^t) does not satisfy integrality requirements (see Sect. 3 for details).
Upper bound. As we have just highlighted, the feasibility check may produce a bilevel feasible solution $(x^t, \hat{y}^t)$, but $\mathcal{R}(x^t) \cap \mathcal{P}_1(x^t)$ may not be a singleton and there may exist a rational reaction $\bar{y} \in \mathcal{R}(x^t) \cap \mathcal{P}_1(x^t)$ to x^t with $d^1\bar{y} < d^1\hat{y}^t$, which results $d^1\hat{y}^t > \Xi(x^t)$ (note that the selected policy is the optimistic one). To compute the "true" objective function value $cx^t + \Xi(x^t)$ associated with x^t and produce the best possible upper bound, we may explicitly compute $\Xi(x^t)$. Once this computation is done and the associated solution and bound are stored, we can safely eliminate x^t from the feasible region by either branching or cutting, as we describe below, in order to force consideration of alternative solutions. This computation is required under certain conditions and may be optionally undertaken in others, depending on the parameter settings described in detail in Sect. 3. $\Xi(x^t)$ is obtained by solving

$$\Xi(x^t) = \min \left\{ d^1 y \;\middle|\; y \in \mathcal{P}_1(x^t) \cap \mathcal{P}_2(x^t) \cap Y, d^2 y \leq \phi(A^2 x^t) \right\},$$

which is an MILP since we have already computed $\phi(A^2x^t)$ in the feasibility check. Solving this MILP to evaluate $\Xi(x^t)$ yields the globally valid upper bound $cx^t + \Xi(x^t)$.

Observe that even the problem of minimizing $\Xi(x)$ over $\{x \in X \mid x_L = x_L^t\}$ is still an MILP and can reveal an upper bound across a range of first-level solutions. This result is formalized in Theorem 1 below.

2.2 Pruning

As is the case with all branch-and-bound algorithms, pruning of node t occurs whenever $L^t \geq U$. This again subsumes two special cases. First, if checking Conditions 3 and 4 verifies that (x^t, y^t) is bilevel feasible, then we must have $L^t = U^t \geq U$. Second, when the relaxation is infeasible, we have $L^t = \infty$ and we can again prune the node.

There is one additional case that is particular to MIBLPs and that is when all linking variables are fixed. In this case, we utilize the property of MIBLPs illustrated in the next theorem.

Theorem 1 [27] *For $\gamma \in \mathbb{Z}^L$, we have*

$$\mathcal{F} \cap \{(x, y) \in X \times Y \mid x_L = \gamma\} = \mathcal{S} \cap \left\{(x, y) \in X \times Y \;\middle|\; d^2 y \leq \phi(A^2 x), x_L = \gamma \right\}.$$

Proof From the definitions of sets $\mathcal{S}$ and $\mathcal{F}$ provided in Sect. 1.1, it follows that

$$\mathcal{F} = \mathcal{S} \cap \left\{(x, y) \in X \times Y \;\middle|\; d^2 y \leq \phi(A^2 x) \right\}.$$

The result follows. □

Corollary 1 *For $\gamma \in \mathbb{Z}^L$, we have*

$$\min\{cx + d^1 y \mid (x, y) \in \mathcal{F}, x_L = \gamma\} = \min\{cx + d^1 y \mid (x, y) \in \mathcal{S}, d^2 y \leq \phi(A^2 x), x_L = \gamma\}. \tag{UB}$$

What Theorem 1 tells us is that when the values of all linking variables are fixed at node t ($l^t_{x_L} = u^t_{x_L}$, either because of branching constraints or otherwise), then optimizing over $\mathcal{F}^t$ is equivalent to optimizing over $\mathcal{S}^t$ while additionally imposing an upper bound on the objective value of the second-level problem. In other words, we have the following.

Corollary 2 *Whenever $\mathcal{F}^t \subseteq \{(x, y) \in \mathcal{F} \mid x_L = \gamma\}$ for some $\gamma \in \mathbb{Z}^L$, then*

$$\min_{(x,y)\in\mathcal{F}^t} cx + d^1 y = \min\left\{cx + d^1 y \;\middle|\; (x, y) \in \mathcal{S}^t, d^2 y \leq \phi(A^2 x), x_L = \gamma\right\}, \tag{UB^t}$$

where $\mathcal{S}^t = \mathcal{P}^t \cap (X \times Y)$.

Therefore, the optimal solution value for the subproblem at node t can be obtained by solving problem (UB^t) with $\gamma = x^t_L$. Furthermore, solving problem (UB) instead of (UB^t) provides a bilevel feasible solution which is at least as good as the optimal solution of node t since it is a relaxation (it is explained in detail in Sect. 4.3 that there may be more than one node with $x_L = \gamma$). Hence, node t can be pruned after solving either (UB^t) or (UB). Note that these problems may be infeasible, in which case node t is infeasible. Although solving problem (UB) may be more difficult than solving problem (UB^t), solving problem (UB) provides useful information about all bilevel feasible solutions with $x_L = x^t_L$ (not only those feasible for node t), so it is generally recommended to solve (UB) instead of (UB^t) (see Sects. 3.2 and 3.3).

Even if the values of linking variables are not fixed at node t ($l^t_{x_L} \neq u^t_{x_L}$), it may be advantageous to solve (UB) with $\gamma = x^t_L$ whenever $x^t_L \in \mathbb{Z}^L$ in order to improve the upper bound. In Sect. 3, we describe the parameters of `MibS` that control when problem (UB) is solved during the solution process. The cases in which, this problem must be solved, regardless of the values of parameters, are also discussed.

2.3 Branching

As previously described, the role of the branching procedure is to remove regions of the feasible set of the relaxation that contain no improving bilevel feasible solutions. This is accomplished by imposing a valid disjunction, which we typically choose such that it is violated by the solution to the current relaxation (i.e., removes it from the feasible region). When the branching procedure is invoked for node t, we have that either

- the solution $(x^t, y^t) \notin \mathcal{F}$ because
  - $(x^t, y^t) \notin X \times Y$ or
  - $d^2 y^t > \phi(A^2 x^t)$ (we have previously solved the corresponding problem (SL-MILP));

  or
- we have $(x^t, y^t) \in X \times Y$ and we are not sure of its feasibility status because we chose not to solve (SL-MILP) (see Sect. 3 for an explanation of the conditions under which we may make this choice).

These alternatives make it clear that we may sometimes want to branch, even though $(x^t, y^t) \in X \times Y$, necessitating a branching scheme that is more general than the one traditionally used in MILP.
Branching on linking variables. Due to Theorem 1, we know that once the values of linking variables are fixed completely at a given node t ($l^t_{x_L} = u^t_{x_L}$), the node can be pruned after solving (UB). One strategy for branching is therefore to think of the search as being over the set of possible values of the linking variables and to prefer a branching disjunction that partitions *this* set of values. In such a scheme, we only consider branching on linking variables as long as any such variables remain unfixed. We show in Sect. 4 that the apparent logic in such a scheme is effective empirically in some cases, but this scheme also raises issues that do not generally arise in branching on variables in MILP. In particular, situations may arise in which there are no linking variables with fractional values ($x^t_L \in \mathbb{Z}^L$) and yet we would still like to (or need to) impose a branching disjunction.

Naturally, when there *does* exist $i \in L$ such that $x^t_i \notin \mathbb{Z}$, we can impose a variable disjunction

$$X_1 = \left\{(x, y) \in \mathbb{R}^{n_1 \times n_2} \;\middle|\; x_i \leq \lfloor x^t_i \rfloor\right\} \quad X_2 = \left\{(x, y) \in \mathbb{R}^{n_1 \times n_2} \;\middle|\; x_i \geq \lfloor x^t_i \rfloor + 1\right\},$$

as usual. When $x^t_L \in \mathbb{Z}^L$, however, it is less obvious what to do. If we either have $x^t_i \notin \mathbb{Z}$ for $k_1 < i \leq r_1$ or we have that $y^t \notin Y$, then there exists the option of breaking with the scheme and imposing a standard variable disjunction on a non-linking variable (those not in set L). Computational experience has shown that this may not always be a good idea empirically.

When $x^t_L \in \mathbb{Z}^L$, there is no single variable disjunction that can be imposed on linking variables that would eliminate x^t from (the projection of) the feasible region of both resulting subproblems. Suppose we nevertheless branch on the variable disjunction associated with some variable indexed $i \in L$. Since $x^t_i \in \mathbb{Z}$ (and assuming that $l^t_{x_i} \neq u^t_{x_i}$), we may branch, e.g., either on the valid disjunction

$$X_1 = \left\{(x, y) \in \mathbb{R}^{n_1 \times n_2} \;\middle|\; x_i \leq x^t_i\right\}, \quad X_2 = \left\{(x, y) \in \mathbb{R}^{n_1 \times n_2} \;\middle|\; x_i \geq x^t_i + 1\right\}$$

or

$$X_1 = \left\{(x, y) \in \mathbb{R}^{n_1 \times n_2} \;\middle|\; x_i \leq x^t_i - 1\right\}, \quad X_2 = \left\{(x, y) \in \mathbb{R}^{n_1 \times n_2} \;\middle|\; x_i \geq x^t_i\right\},$$

preferring the former when $x_i^t \leq u_{x_i}^t - 1$ (otherwise, one of the resulting subproblem will be trivially infeasible and the other will be equivalent to node t). This means x^t satisfies either the first or second term of this disjunction. Although imposing this valid disjunction seems undesirable, since it does not remove (x^t, y^t) from the union of the feasible regions of the resulting subproblems, let us explore the logic of the approach.

Suppose we continue to branch in this way and consider the indices of the set of leaf nodes T^t of the subtree of the search tree rooted at node t. Since this set of leaf nodes represents a partition of $\mathcal{S}^t$, the projection of the feasible region of exactly one of these nodes contains x^t. We have that the values of all linking variables are fixed at this node, since we would otherwise continue branching (we have already noted that such node can be pruned after solving problem (UB)). The projection of the union of the feasible regions of the remaining leaf nodes is thus equal to

$$\operatorname{proj}_x(\mathcal{S}^t) \setminus \left\{x \in X \;\middle|\; x_L = x_L^t\right\}.$$

Hence, this branching rule can be seen as implicitly branching on the disjunction

$$X_1 = \left\{(x, y) \in \mathbb{R}^{n_1 \times n_2} \;\middle|\; x_L = x_L^t\right\}, \quad X_2 = \left\{(x, y) \in \mathbb{R}^{n_1 \times n_2} \;\middle|\; x_L \neq x_L^t\right\},$$

which is obviously valid.

As a practical matter, if we implement the scheme of only branching on linking variables using simple variable disjunctions, we inevitably create a number of additional nodes for which x^t remains in the projection of the feasible set (one at each level of the tree down to the leaf node at which all linking variables are fixed). This can easily be detected using the scheme for maintaining a pool of linking solutions that have already been seen and will not cause any significant computational issues. For the remaining nodes, we are guaranteed that x^t is not contained in the projection (though these nodes may also be the root of a subtree in which all linking variables are integer-valued). An alternative would be to branch in a non-binary way using a more complicated disjunction that would directly create the leaf nodes of the subtree rooted at node t, but there seems to be no advantage to such a scheme.

In forming the list of variables that are candidates for branching, we add only linking variables with fractional values if $x_L^t \notin \mathbb{Z}^L$. If $x_L^t \in \mathbb{Z}^L$, then we are forced to add integer-valued linking variables that remain unfixed to the list. If all linking variables are fixed, then we have several options, depending on how parameters are set (see Sect. 3.2).

- We may prune the node after solving problem (UB), as explained in Sect. 2.2.
- When $(x^t, y^t) \notin X \times Y$, we may add non-linking variables with fractional values to the candidate set.
- (x^t, y^t) may be removed by generating a cut (after solving (SL-MILP) if $(x^t, y^t) \in X \times Y$, see Sect. 2.4).

Branching on fractional variables. Naturally, we may also consider a more traditional branching scheme in which we only branch on variables with fractional values. In such a scheme, we must consider branching on all variables (first- and second-level), since we may have no other option when $x^t \in X$.

Selecting candidates. We have so far neglected to address the question of *which* variable to choose as the basis for our branching disjunction when there is more than one candidate. The idea of limiting branching only to the linking variables differs from the usual scheme employed in algorithms for MILP in that we only consider a subset of the integer variables for branching. Nevertheless, in both of the schemes described above, we face a similar decision regarding which of the (possibly many) variables available for branching should actually be chosen. It is well-known that the choice of branching variable can make a substantial difference in practical performance.

In branching procedures for MILP, it is typical to first choose a set of candidates for branching (typically, all integer variables with fractional values are considered). The selection of the "best" branching candidate is then made from this set of candidates based on one of a number of schemes for scoring/predicting the "impact" of choosing a particular branching variable. The details of these scoring mechanisms can be found in any number of references (see, e.g., [36]). Although the method used to define the set of branching candidates differs from the one used in MILP, the method employed in `MibS` for selecting from the candidates is similar to the schemes that can be found in the literature. `MibS` offers the pseudocost, strong branching and reliability branching schemes.

2.4 Cutting

The generation of valid inequalities is an alternative to branching for removing infeasible and/or non-improving solutions. We refer the reader to [37,38] for theoretical background on the separation problem and cutting plane methods in general. In the case of MIBLP, the valid inequalities we aim to generate are those that separate the solution to the current relaxation from the convex hull of the (improving) bilevel feasible solutions. These inequalities are generated iteratively in order to improve the relaxation, as in a standard cutting plane method. DeNegre and Ralphs [19] showed that such a pure cutting plane algorithm can be used to solve certain classes of MIBLPs.

In order to allow the separation of points that are in $\text{conv}(\mathcal{F})$ but have already been identified (or can be shown not to be improving), we recall here the notion of improving valid inequality that we introduced in Definition 3, re-stating it in the notation of (MIBLP-VF). A triple $(\alpha^x, \alpha^y, \beta) \in \mathbb{R}^{n_1+n_2+1}$ is said to constitute an *improving valid inequality* for $\mathcal{F}$ if

$$\alpha^x x + \alpha^y y \geq \beta \ \forall (x, y) \in \text{conv}\left(\left\{(x, y) \in \mathcal{F} \ \middle|\ cx + d^1 y < U\right\}\right),$$

where U is the current global upper bound. As mentioned before, we shall drop the word "improving" from the remaining discussion.

In Sect. 2.1, we stated that the difference between the feasible region of a subproblem and that of the original problem is only in changes to the bounds on variables. This is no longer strictly true when valid inequalities are also being generated and added dynamically to the constraint set of the relaxation. The feasible region $\mathcal{P}^t$ of the relaxation at node t from (LR^t) then becomes

$$\mathcal{P}^t = \left\{(x, y) \in \mathcal{P} \cap \Pi^t \;\middle|\; l_x^t \leq x \leq u_x^t, l_y^t \leq y \leq u_y^t\right\},$$

where Π^t is a polyhedron representing the valid inequalities applied at node t. The set of valid inequalities may include inequalities generated at any of the ancestor nodes in the path to the root node of the search tree.

There are several distinct categories of valid inequality that can be generated, depending on the feasibility conditions violated by the solution (x^t, y^t) that we are trying to separate at node t. Generally speaking, all valid inequalities can be classified as

– Feasibility cuts: Inequalities valid for conv $\left\{(x, y) \in \mathcal{S} \;\middle|\; cx + d^1 y < U\right\}$.
– Optimality cuts: Inequalities valid for conv $\left\{(x, y) \in \mathcal{F} \;\middle|\; cx + d^1 y < U\right\}$.
– Projected optimality cuts: Inequalities valid for conv$\{(x, y) \in \mathbb{R}^{n_1 \times n_2} \mid x \in \mathcal{F}_1, cx + \Xi(x) < U\}$.

The feasibility cuts include those classes generally employed in solution of MILPs. Inequalities used to remove solutions $(x^t, y^t) \in \mathcal{S}$ at node t may be referred to roughly as *optimality cuts*, while *projected optimality cuts* are those that may remove all solutions with specific first-level values. For example, inequalities that remove the solutions with the same linking component as x^t once the problem (UB) with $\gamma = x_L^t$ has been solved. Clearly, these three classes are not entirely distinct and the categorization is meant only to provide a rough categorization.

`MibS` includes separation routines for a variety of known classes of valid inequalities in the current literature. We give a brief overview here. More details are provided in a companion paper that analyzes the impact of various classes of inequalities in greater detail.

Integer no-good cut. This class of inequalities are valid for problems in which $r_1 = n_1$, $r_2 = n_2$ and all problem data are integer (with the possible exception of the objective function). It was introduced by DeNegre and Ralphs [19] and is derived from a split disjunction obtained by taking combinations of inequalities binding at (x^t, y^t) in a fashion similar to that used in deriving the Chvátal cuts valid for the feasible regions of MILPs. The cut eliminates a single extremal solution that is integral yet not bilevel feasible from the feasible region of the relaxation. It is easy to derive such a cut under the given assumptions.

Generalized no-good cut. This class of inequalities are valid for problems in which all linking variables are binary. It is a generalization of the *no-good cut* introduced by DeNegre [10] in the context of MIBLPs. Similar cuts have been used in many other contexts. The idea of a "no-good cut" seems to have first been suggested by Balas and Jeroslow [39]. The purpose of this cut is to remove all solutions for which the linking variables have certain fixed values. If at node t, we have $x_L^t \in \mathbb{Z}^L$ and either (i) we choose to solve (SL-MILP) and it is infeasible or (ii) we choose to solve (UB), then we can store x_L^t in the linking solution pool (if applicable) and add a generalized no-good cut with $\gamma = x_L^t$ in order to avoid generating the same linking solution again. This inequality can also be added in other subproblems if/when this linking solution arises again.

Intersection cut. Fischetti et al. [27,28] generalized the well-known concept of an intersection cut [40], used extensively in the MILP setting, to MIBLPs. The various classes introduced differ from each other in the way the underlying "bilevel-free" convex set used for generating the cuts is defined. Three classes of introduced cuts are working for the problems with $G^2y - A^2x \in \mathbb{Z}^{m_2}$ for all $(x, y) \in \mathcal{S}$ and $d^2 \in \mathbb{Z}^{n_2}$. These cuts can be employed for removing the infeasible solution $(x^t, y^t) \notin \mathcal{F}$ from the feasible region, but it is not guaranteed when $(x^t, y^t) \notin \mathcal{S}$. The other class known as *hypercube intersection cut* is valid for the MIBLPs with $x_L \subseteq \mathbb{Z}^L$ and can be added when $x_L^t \in \mathbb{Z}^L$. We solve (UB) and store the solution in the linking solution pool (if applicable) and after doing so, the hypercube intersection cut can be added. This inequality is guaranteed to be violated by (x^t, y^t) and *may* also eliminate *some* other solutions for which $x_L = x_L^t$, but is not guaranteed to eliminate all such solutions.

Increasing objective cut. This class of inequalities are valid for problems in which linking variables are binary and $A^2 \geq 0$. Proposed by DeNegre [10], it is a disjunctive cut derived from the following valid disjunction based on the value function bound (VFB).

$$X_1 = \left\{(x, y) \in \mathbb{R}^{n_1 \times n_2} \;\middle|\; \sum_{i \in L : x_i^t = 0} x_i = 0, d^2 y \leq d^2 \hat{y}\right\},$$

$$X_2 = \left\{(x, y) \in \mathbb{R}^{n_1 \times n_2} \;\middle|\; \sum_{i \in L : x_i^t = 0} x_i \geq 1\right\},$$

where $\hat{y} \in \mathcal{R}(x^t)$. In addition to the way in which this disjunction was exploited by DeNegre [10], there are other inequalities that could also be derived from this disjunction. This cut can be applied to eliminate $(x^t, y^t) \in X \times Y$ from the feasible region at node t when (x^t, y^t) is found to be infeasible. Note that by solving (UB) following the feasibility check, we could also apply the stronger generalized no-good cut in this situation.

Benders cut. This class of inequalities are valid for problems in which linking variables are binary and the second-level variables coefficients are not greater than 0 in the second-level constraints in which the linking variables do not participate. Furthermore, corresponding to each linking variable x_i, there exists y_i so that $x_i = 1$ results $y_i = 0$ and this is the only restriction from the second-level constraints in which the linking variables participate. This cut was originally mentioned by Caprara et al. [23] in the context of knapsack interdiction problems, but is valid for a broader class of problems. This cut utilizes the value function bound (VFB) and a new such cut can be imposed anytime we find a new feasible solution.

2.5 Primal heuristics

Just as in the solution of MILPs, the primal heuristics can be employed within a branch-and-cut algorithm for solving MIBLPs in order to enhance the discovery of bilevel feasible solutions. Three different heuristics introduced in [10] are implemented in `MibS`, as follows.

Improving objective cut heuristic. Let $(\bar{x}, \bar{y}) \in \mathcal{S}$ and $\hat{y} \in \mathcal{R}(\bar{x})$. As we discussed in Sect. 2.1, $(\bar{x}, \hat{y})$ is a bilevel feasible solution if it satisfies the first-level constraints. However, $(\bar{x}, \hat{y})$ is not necessarily a good solution with respect to the first-level objective function. The idea of this heuristic is to exploit the information from both $(\bar{x}, \bar{y})$ (which is a good solution with respect to the first-level objective) and $(\bar{x}, \hat{y})$ (which is a likely bilevel feasible solution). To do so, we first determine

$$(\tilde{x}, \tilde{y}) \in \operatorname{argmin} \left\{ cx + d^1 y \;\middle|\; (x, y) \in \mathcal{S}, d^2 y \leq d^2 \hat{y} \right\}.$$

When $(\bar{x}, \hat{y}) \in \mathcal{F}$, such $(\tilde{x}, \tilde{y})$ must exist. Further, we must have $(\tilde{x}, \tilde{y}) \in \mathcal{F}$. A separate feasibility check is thus required and this check is what is expected to finally produce the (potentially) improved solution.

Second-level priority heuristic. This heuristic is similar to the improving objective cut heuristic in that it also tries to balance bilevel feasibility with the quality of obtained solution. The MILP solved with this heuristic, however, is

$$\min \left\{ d^2 y \;\middle|\; (x, y) \in \mathcal{S}, cx + d^1 y \leq U \right\}, \tag{PH}$$

where U is the current upper bound. In this problem, the goal of the added constraint is to improve the quality of the solution, while the objective function of this problem increases the chance of generating a bilevel feasible solution (however, it does not guarantee the bilevel feasibility of the produced solution). Note that the upper bound U could be replaced by any chosen "target" to try to ensure improvement on the current incumbent, but then we would not be assured feasibility of (PH).

Weighted sums heuristic. This heuristic employs some techniques from multi-objective optimization to generate bilevel feasible solutions. The main idea of this heuristic is finding a subset of *efficient* solutions (those for which we cannot improve one of the objectives without degrading the other while maintaining feasibility [41]) of the following problem by using a weighted-sum subproblem [42].

$$\operatorname*{argvmin}_{(x,y)\in\mathcal{S}} \{cx + d^1 y, d^2 y\},$$

where the operator argvmin means finding a solution (or more than one) that is efficient. For more details, see [10].

3 Software framework

In this section, we describe the implementation of the `MibS` software [43]. This paper refers to version `1.1.2`, the latest released version at the time of this writing. The overall algorithm employed in `MibS` combines the procedures described in Sect. 2 in a fashion typical of existing branch-and-cut algorithms for other problems. In fact, MibS is built on top of BLIS, a parallel implementation of branch and cut for the solution of MILPs [44]. In Sect. 3.1, we describe the class structure of the C++ code that encapsulates the implementation. Furthermore, there are a number of important parameters, described in Sect. 3.2, that determine how the various components described in Sect. 2 are coordinated. Although there are defaults that are set automatically after analyzing the structure of a particular instance, these parameters can and should be tuned for use in particular applications, Understanding their role in the algorithm is crucial to understanding the overall strategy, described in Sect. 3.3. Finally, in Sect. 3.4, we describe important implementational issues surrounding how the second-level problem is solved.

3.1 Design of MibS

We first briefly describe the overall design of the software. MibS is an open-source implementation in C++ of the branch-and-cut algorithm described in Sect. 3.3. In addition to a core library, MibS utilizes a number of other open-source packages available from the Computational Infrastructure for Operations Research (COIN-OR) repository [2]. These packages include the following.

- The COIN-OR High Performance Parallel Search (CHiPPS) Framework [45], which includes a hierarchy of three libraries.
  - Abstract Library for Parallel Search (ALPS) [46,47]: This project is utilized for managing the global branch and bound.
  - Branch, Constrain, and Price Software (BiCePS) [45,48]: This project provides base classes for objects used in MibS.
  - BiCePS Linear Integer Solver (BLIS) [44,49]: This project is the parallel MILP Solver framework from which MibS is derived.
- COIN-OR Linear Programming Solver (CLP) [50]: MibS employs this software for solving the LPs arising in the branch-and-cut algorithm.
- SYMPHONY [51,52]: This software is used for solving the MILPs required to be solved in the branch-and-cut algorithm.
- COIN-OR Cut Generation Library (CGL) [53]: MibS and SYMPHONY utilize this library to generate valid inequalities valid for MILPs.
- COIN-OR Open Solver Interface (OSI) [54]: This project is used for interfacing with solvers, such as SYMPHONY, Cbc and CPLEX.

MibS is comprised of a number of classes that are either specific to MibS, or are classes derived from a class in one of the libraries of CHiPPS. The main classes of MibS are as follows.

- MibSModel: This class is derived from the BlisModel class. It extracts and stores the model from the input files, which consist of an MPS file and an auxiliary information file. We refer the reader to the README file of MibS for a comprehensive description of the input file format.
- MibSBilevel: This class is specific to MibS and is utilized for checking the bilevel feasibility of solutions of the relaxation problem. Furthermore, this class

finds the bilevel feasible solutions, as described in lines 18, 23 and 28 of Algorithm 1.

- `MibSTreeNode`: This class is derived from the class `BlisTreeNode` and contains the methods for processing the branch-and-bound nodes.
- `MibSCutGenerator`: This class is specific to `MibS` and contains the methods for generating the valid inequalities described in Sect. 2.4.
- `MibSBranchStrategyXyz`: These classes (one for each strategy `MibS` can employ for selecting the final branching candidate: `Pseudo`, `Strong`, and `Reliability`) are derived from the parent classes `BlisBranchStrategyXyz` and contain the implementation used for selecting the final branching candidate.
- `MibSHeuristic`: This class is specific to `MibS` and contains the methods for generating heuristic solutions by employing the primal heuristics illustrated in Sect. 2.5.
- `MibSSolution`: This is a class derived from `BlisSolution` class and is utilized for storing the bilevel feasible solutions.

Since one important feature of a practical solver is ease of use, we have tried to make `MibS` as user-friendly as possible. Prior to the solution process, `MibS` analyzes the problem to determine its properties, e.g., the type of instance (interdiction or general), the type of variables present at each level (continuous, discrete, or binary) and signs of the coefficients in the constraint matrices. It then checks the parameters set by the user and modifies them if it determines that those values of parameters are not valid for this problem, informing the user of any change in the parameters. For example, `MibS` turns off any cuts selected by the user that are not valid for the instance to be solved.

3.2 Parameters

A wide range of parameters are available for controlling the algorithm. *Branching Strategy.* There are several parameters that control the branching strategy. The main one is `branchStrategy`, which controls what variables we allow as branching candidates. The options are

- `linking`: Branch only on linking variables, as long as any such variable remains unfixed, with priority given to such variables with fractional values.
- `fractional`: Branch on any integer first- or second-level variable that has fractional value, as is traditional in solving MILPs.

We also have parameters for controlling the strategy by which the final branching candidate is selected (`strong, pseudocost, reliability`). The default is to use `pseudocost` scoring.

Search strategy. The search strategy used by `MibS` is controlled by ALPS, the underlying tree search framework. The default is the best-first strategy.

Cutting strategy. There are parameters for the types of valid inequalities to generate and the strategy for when to generate them (only in the root node, periodically, etc.). Note that we are forced to generate cuts whenever there are no available branching candidates and the current node cannot be pruned. With `branchStrategy` set to `linking`, the parameters for solving problems (SL-MILP) and (UB) (which are described later) can be set so that a pure branch and bound is possible, but this is not possible for the `fractional` case. The default settings for cuts depends on the instance. `MibS` includes an automatic analyzer that determines which classes of cuts are applicable and likely to be effective for a given instance. The frequency of generation is selected automatically by default, based on statistics gathered during early stages, as is standard in many solvers.

Primal heuristics. Types of primal heuristics to employ and the strategy for how often to employ them (see Sect. 2.5). Only BLIS (generic MILP) heuristics are turned on by default.

Linking solution pool. There is a parameter `useLinkingSolutionPool` that determines whether to maintain a pool $\mathcal{L}$ of linking solutions seen so far, as described earlier, in order to avoid solving identical instances of (SL-MILP) and (UB). When the parameter is set to `True`, we check $\mathcal{L}$ before solving either of (SL-MILP) or (UB).

Each linking solution stored in $\mathcal{L}$ is stored along with both a status tag and, if appropriate, an associated solution. The status tag is one of the following:

- `secondLevelIsInfeasible`: If the corresponding problem (SL-MILP) is solved and it is infeasible.
- `secondLevelIsFeasible`: If the corresponding problem (SL-MILP) is solved and it has an optimal solution, but problem (UB) is not solved.
- `UBIsSolved`: If the corresponding problem (UB) is solved.

All linking solutions are stored in a hash table in order to enable an efficient membership check.

Feasibility check. The following binary parameters determine the strategy for when to solve the second-level problem (SL-MILP) (for details on solution of the second-level problem, see Sect. 3.4).

- `solveSecondLevelWhenLVarsFixed`: Whether to solve when $l^t_{x_L} = u^t_{x_L} = x^t_L$ (linking variables fixed).
- `solveSecondLevelWhenLVarsInt`: Whether to solve when $x^t_L \in \mathbb{Z}^L$.
- `solveSecondLevelWhenXVarsInt`: Whether to solve when $x^t \in X$.
- `solveSecondLevelWhenXYVarsInt`: Whether to solve when $(x^t, y^t) \in X \times Y$.

When problem (SL-MILP) is solved, we have the following implications, depending on the result.

- If (SL-MILP) is infeasible, then all solutions (x, y) with $x_L = x^t_L$ are infeasible and can be removed from the feasible region of the relaxation either by generation of a cut or by branching. To avoid solving problems (SL-MILP) for a different solution with the same linking part, the tuple $[x^t_L$, `secondLevelIsInfeasible`$]$ can be added to the linking solution pool $\mathcal{L}$.
- If (SL-MILP) has an optimal solution, we can avoid solving (SL-MILP) for any (x, y) for which $x_L = x^t_L$ arising in the future by adding the tuple

[x^t_L, `secondLevelIsFeasible`] and the associated solution to (SL-MILP) to the linking solution pool $\mathcal{L}$.

Regardless of parameter settings, we must always solve (SL-MILP) whenever $(x^t, y^t) \in X \times Y$, $x^t_L \notin \mathcal{L}$ (we have not previously solved (SL-MILP) for x^t_L), and there are no branching candidates. Clearly, if we have branching candidates, then we can avoid solution of (SL-MILP) by branching. Similarly, if $(x^t, y^t) \notin X \times Y$, we can generate standard MILP cuts. Otherwise, we must have either

– `branchStrategy` is `fractional`, in which case we must solve (SL-MILP) and then may either remove (x^t, y^t) by generating a valid inequality (if infeasible) or prune the node (if feasible).
– `branchStrategy` is set to `linking` and all linking variables are fixed, in which case we must also solve (UB) (if (SL-MILP) is feasible) and prune the node.

Computing best UB. The following binary parameters determine when the problem (UB) should be solved in order to compute the best bilevel feasible solution (x, y) with $x_L = x^t_L$ (if such solution exists).

– `computeBestUBWhenLVarsFixed`: Whether to solve when $l^t_{x_L} = u^t_{x_L} = x^t_L$.
– `computeBestUBWhenLVarsInt`: Whether to solve when $x^t_L \in \mathbb{Z}^L$.
– `computeBestUBWhenXVarsInt`: Whether to solve when $x^t \in X$.

After solving (UB), we know that no solution (x, y) with $x_L = x^t_L$ can be improving and thus, all such solutions can be removed either by generation of a cut or by branching. To avoid solving problem (UB) for any subsequent solutions (x, y) for which $x_L = x^t_L$, the tuple [x^t_L, `secondLevelIsFeasible`] should be replaced with the tuple [x^t_L, `UBIsSolved`] and the associated solution stored in the linking solution pool $\mathcal{L}$.

Note that it would be possible to solve (UB^t) rather than (UB) if the goal were only to prune the node. Solving (UB) (which may not be much more difficult) allows us to exploit the solution information globally through the linking solution pool.

Regardless of parameter setting, we must always solve problem (UB) (if it has not been previously solved for x^t_L) when (i) `branchStrategy` is `linking`, (ii) $(x^t, y^t) \in X \times Y$ and is bilevel infeasible and (iii) all linking variables are fixed. This does not mean that in this case, solving problem (UB) (and further fathoming node t) is the only algorithmic option, as explained in Sect. 2.3.

3.3 Outline of the algorithm

Algorithm 1 gives the general outline of the node processing loop in this branch-and-cut algorithm. Note that for simplicity, the algorithm as stated assumes that the linking pool is used, though the option of not using this pool is also provided. In almost all cases, use of the linking pool is advantageous (see Sect. 4.3). At a high level, the procedure consists of the following steps.

1. Solve the relaxation (LR^t) (line 3) and prune the node (lines 5–6) if either

Algorithm 1: Node processing loop in `MibS`

```
Input  : [Set $\mathcal{L}$, $U$]
Output: [Set $\mathcal{L}$, $U$, $L^t$, $U^t$]
1  branch ← False, $L^t \leftarrow -\infty$, $U^t \leftarrow \infty$
2  while branch is False do
3      Solve (LR$^t$)
4      $L^t \leftarrow$ The optimal value of (LR$^t$)
5      if (LR$^t$) is infeasible or $L^t \geq U$ or
       ($l^t_{x_L} = u^t_{x_L}$ and ([$x^t_L$, secondLevelIsInfeasible] $\in \mathcal{L}$ or [$x^t_L$, UBIsSolved] $\in \mathcal{L}$))
       then
6          Fathom node $t$
7      if [$x^t_L$, $\cdot$] $\notin \mathcal{L}$ and
        ((branchStrategy is linking and $(x^t, y^t) \in X \times Y$ and $l^t_{x_L} = u^t_{x_L}$) or
         (branchStrategy is fractional and $(x^t, y^t) \in X \times Y$) or
         (solveSecondLevelWhenXYVarsInt and $(x^t, y^t) \in X \times Y$) or
         (solveSecondLevelWhenXVarsInt and $x^t \in X$) or
         (solveSecondLevelWhenLVarsInt and $x_L \in \mathbb{Z}^L$)or
         (solveSecondLevelWhenLVarsFixed and $l^t_{x_L} = u^t_{x_L}$)) then
8          Solve (SL-MILP) to find $\phi(A^2 x^t)$
9          if (SL-MILP) is infeasible then
10             $\mathcal{L} \leftarrow \mathcal{L} \cup$ [$x^t_L$, secondLevelIsInfeasible]
11             if $l^t_{x_L} = u^t_{x_L}$ then
12                 Fathom node $t$
13         else
14             $\mathcal{L} \leftarrow \mathcal{L} \cup$ [$x^t_L$, secondLevelIsFeasible]
15     if [$x^t_L$, secondLevelIsFeasible] $\in \mathcal{L}$ or [$x^t_L$, UBIsSolved] $\in \mathcal{L}$ then
16         $\hat{y}^t \leftarrow$ The optimal solution of (SL-MILP)
17         if $(x^t, y^t) \in \mathcal{F}$ then
18             $U^t \leftarrow cx^t + d^1 y^t$, $U \leftarrow \min\{U, U^t\}$
19             Fathom node $t$
20         if [$x^t_L$, UBIsSolved] $\notin \mathcal{L}$ and
            ((branchStrategy is linking and $(x^t, y^t) \in X \times Y$ and $l^t_{x_L} = u^t_{x_L}$) or
             (computeBestUBWhenXVarsInt and $x^t \in X$) or
             (computeBestUBWhenLVarsFixed and $l^t_{x_L} = u^t_{x_L}$) or
             (computeBestUBWhenLVarsInt)) then
21             Solve (UB)
22             if (UB) is feasible then
23                 $U \leftarrow \min\{U$, optimal value of (UB)$\}$
24             $\mathcal{L} \leftarrow \mathcal{L} \setminus$ [$x^t_L$, secondLevelIsFeasible], $\mathcal{L} \leftarrow \mathcal{L} \cup$ [$x^t_L$, UBIsSolved]
25             if $l^t_{x_L} = u^t_{x_L}$ then
26                 Fathom node $t$
27         else if $(x^t, \hat{y}^t) \in \mathcal{F}$ then
28             $U \leftarrow \min\{U, cx^t + d^1 \hat{y}^t\}$
29     if branchStrategy is fractional and $(x^t, y^t) \in X \times Y$ then
30         Remove $(x^t, y^t)$ by generating a cut
31     else if [$x^t_L$, $\cdot$] $\notin \mathcal{L}$ and $(x^t, y^t) \in X \times Y$ then
32         branch ← True
33     else
34         Remove $(x^t, y^t)$ by generating a cut or branch ← True
```

- (LR^t) is infeasible;
- $L^t \geq U$; or
- x_L is fixed and it has been stored in set $\mathcal{L}$ with either `UBIsSolved` or `secondLevelIsInfeasible` tags.

2. If (SL-MILP) was not previously solved with respect to x_L^t, then depending on (x^t, y^t) and the parameter settings, we may next solve it (lines 7–8).
   - (SL-MILP) is infeasible $\Rightarrow x^t \notin \mathcal{F}_1$ (line 9)
     - Add $[x_L^t,$ `secondLevelIsInfeasible`] to $\mathcal{L}$ (line 10).
     - If x_L is fixed, fathom node t (lines 11–12).
   - (SL-MILP) is feasible $\Rightarrow$ add $[x_L^t,$ `secondLevelIsFeasible`] to $\mathcal{L}$ (lines 13–14).
3. If (SL-MILP) was solved now or previously and is feasible (line 15), we have either
   - $(x^t, y^t) \in \mathcal{F} \Rightarrow$ update U and fathom node t (lines 17–19).
   - $(x^t, y^t) \notin \mathcal{F}$
     - Update U if $(x^t, \hat{y}^t) \in \mathcal{F}$ (in case of not solving (UB)) (lines 27–28) and eliminate (x^t, y^t) (lines 29–34).
     - If (UB) was not previously solved with respect to x_L^t, then depending on (x^t, y^t) and the parameter settings, we may solve it (lines 20–21).
       - (UB) is feasible $\Rightarrow$ update U (lines 22–23).
       - Remove $[x_L^t,$ `secondLevelIsFeasible`] from set $\mathcal{L}$ and add $[x_L^t,$ `UBIsSolved`] (line 24).
       - If x_L is fixed, fathom node t (lines 25–26).
4. Finally, we must either branch or remove (x^t, y^t) by adding valid inequalities (lines 29–34).
   - If there are no branching candidates, then we must remove (x^t, y^t) by adding valid inequalities (lines 29–30).
   - If (SL-MILP) was not solved now or previously, we must branch if $(x^t, y^t) \in X \times Y$ (lines 31–32).
   - Otherwise, we have the choice of either adding valid inequalities or branching (lines 33–34).

In the case of not using the linking solution pool, the structure of algorithm is similar to Algorithm 1, but differs in the steps where information from the linking solution pool is exploited or new information is added to this pool. For example, the lines 10, 14 and 24 are eliminated and the lines 5, 7, 15 and 20 are modified.

As mentioned earlier, this algorithm can be adapted for other risk functions. For example, the pessimistic risk function (RF-PES) can be accommodated with a few modifications as follows. The same relaxation is used, but the feasibility check is slightly different. At node t, if $(x^t, y^t) \notin X \times Y$, it is infeasible and should be removed, as usual. Otherwise, we compute $\phi(A^2x^t)$ and check whether $d^2y^t = \phi(A^2x^t)$. If $d^2y^t > \phi(A^2x^t)$, (x^t, y^t) is infeasible, as in the optimistic case, but $d^2y^t = \phi(A^2x^t)$ does not guarantee its feasibility. Feasibility of (x^t, y^t) must be verified by also ensuring that d^1y^t is equal to the optimal value of the MILP

$$\max \left\{ d^1 y \;\middle|\; y \in \mathcal{P}_1(x^t) \cap \mathcal{P}_2(x^t) \cap Y, d^2 y \le \phi(A^2 x^t) \right\}.$$

Finding the best feasible solution with $x_L = \gamma \in \mathbb{Z}^L$ requires solving a new bilevel problem and its details are beyond the scope of this paper. To avoid this, the `fractional` branching strategy can be used instead.

3.4 Solving the second-level problem

It is evident that in most cases, much (if not most) of the computational effort in executing the algorithm arises from the time required to solve (SL-MILP) and (UB) (see Table 2). A major focus of ongoing work, therefore, is the development of methodology to reduce the time spent solving these problems.

In closely related work on solving two-stage stochastic mixed integer optimization problems, methodology for warm-starting the solution process of an MILP using information derived from previously solved instances has been developed. Hassanzadeh and Ralphs [55] described a method for solving a sequence of MILPs differing only in the right-hand side. It is shown that a sequence of such solves can be performed within a single branch-and-bound tree, with each solve starting where the previous one left off. Under mild conditions, this method can be used to construct a complete description of the value function ϕ. This method of solving such a sequence of MILPs has been implemented within the SYMPHONY MILP solver [51,52,56], which is one of several supported solvers that can be used for solution of (SL-MILP) and (UB) within `MibS`. Other supported solvers include CPLEX [57] and Cbc [58]. The parameter `feasCheckSolver` determines which MILP solver to be employed.

4 Computational results

A number of experiments were conducted to evaluate the impacts of the various algorithmic options provided by `MibS`. The parameters investigated in this section are

- The parameters that determine when the problems (SL-MILP) and (UB) should be solved.
- The parameter that determines the branching strategy.
- The parameter that determines whether to use the linking solution pool or not.
- The parameters that determine whether to use the primal heuristics or not.

Because of space constraints and to allow more in-depth analysis, testing of the effectiveness of various classes of valid inequalities and parameters for controlling them is not included here, but will be the subject of a future study. Three different data sets (171 instances in total) were employed in our experiments as follows.

- `IBLP-DEN`: This set was generated by DeNegre [10], and contains 50 instances with 15–20 integer variables and 20 constraints, all at the second level.
- `IBLP-FIS`: This is a selected set of 21 of the instances generated by Fischetti et al. [27]. These instances originate from `MILPLIB 3.0` [59] and all variables are binary and all constraints are at the second level.
- `MIBLP-XU`: This set was introduced by Xu and Wang [20] and includes 100 randomly-generated instances. In these problems, all first-level variables are inte-

Table 1 The summary of data sets

Data set	First-level Vars Num	Second-level Vars Num	First-level Cons Num	Second-level Cons Num	First-level Vars Type	Second-level Vars Type	Size
IBLP-DEN	5–15	5–15	0	20	Discrete	Discrete	50
IBLP-FIS	4–2481	2–2480	0	16–4944	Binary	Binary	21
MIBLP-XU	10–460	10–460	4–184	4–184	Discrete	Continuous, discrete	100

ger with upper bound 10, while the second-level variables are continuous with probability 0.5. The number of first- and second-level variables are equal and n_1 is in the range of 10–460 with an increments of 50. Furthermore, the number of first-level and second-level constraints are equal to $0.4n_1$. To have specific bounds on all integer variables, we added the very loose upper bound 1500 for all integer second-level variables in converting these instances to the `MibS` format.

Table 1 summarizes the properties of the data sets. Note that in the described data sets, all first-level variables are linking.

All computational results we report were generated on compute nodes running the Linux (Debian 8.7) operating system with dual AMD Opteron 6128 processors and 32 GB RAM and all experiments were run sequentially. The time limit was 3600 s and the pseudocost branching strategy was used to choose the best variable among the branching candidates for all experiments. In all numerical experiments, the generation of generic MILP cuts by the Cut Generation Library [53] was disabled, since these cuts seem unlikely to be effective in addition to the classes specific to MIBLPs and their integration can cause other algorithmic issues that would first need to be addressed. SYMPHONY was employed as the MILP solver (preprocessing and primal heuristics were turned off) in all experiments, unless otherwise noted. In all experiments, the integer no-good cut was employed for solving the instances of `IBLP-DEN` and `IBLP-FIS` sets, and the problems belonging to the `MIBLP-XU` set were solved by using the hypercube intersection cut. Furthermore, all primal heuristics of `MibS` were disabled in the numerical experiments except as otherwise noted, since these have in general also not proven to be very effective.

All instances were initially solved by all methods described in Sects. 4.1–4.3 below, but in plotting performance profiles, we chose the 125 problems that could be solved by at least one method in 3600 s and whose solution time exceeds 5 s for at least one method. This test set was used for all plots and tables shown in Sect. 4. The details of the results employed for plotting all of these figures and tables are shown in an online supplement (the reported running times do not include the time for reading the instances).

4.1 Impact of strategy for solving (SL-MILP) and (UB)

In order to assess the effect of the parameters for solving problems (SL-MILP) and (UB), we employed five different methods:

- `whenLInt-LInt`: Problems (SL-MILP) and (UB) were both solved only when $x_L^t \in \mathbb{Z}^L$, i.e., the parameters
  - `solveSecondLevelWhenLVarsInt` and
  - `computeBestUBWhenLVarsInt`

  were set to `true`.
- `whenLInt-LFixed`: Problem (SL-MILP) was solved when $x_L^t \in \mathbb{Z}^L$ and (UB) was solved when all linking variables are fixed, i.e., the parameters
  - `solveSecondLevelWhenLVarsInt` and
  - `computeBestUBWhenLVarsFixed`

  were set to `true`.
- `whenLFixed-LFixed`: Problems (SL-MILP) and (UB) were solved only when linking variables were fixed, i.e., the parameters
  - `solveSecondLevelWhenLVarsFixed` and
  - `computeBestUBWhenLVarsFixed`

  were set to `true`.
- `whenXYInt-LFixed`: Problem (SL-MILP) was solved only when $(x^t, y^t) \in X \times Y$ and problem (UB) was solved only when, in addition, all linking variables were fixed, i.e., the parameters
  - `solveSecondLevelWhenXYVarsInt` and
  - `computeBestUBWhenLVarsFixed`

  were set to `true`.
- `whenXYIntOrLFixed-LFixed`: Problem (SL-MILP) was only solved when $(x^t, y^t) \in X \times Y$ or all linking variables were fixed and problem (UB) was solved only whenever all linking variables are fixed, i.e., the parameters
  - `solveSecondLevelWhenXYVarsInt`,
  - `solveSecondLevelWhenLVarsFixed`, and
  - `computeBestUBWhenLVarsFixed`

  were set to `true`.

In this first set of experiments, the `branchStrategy` was set to `linking` and `useLinkingSolutionPool` was set to `true`. The performance profiles shown in Fig. 2 compare the solution time of the five described methods.

Figure 2a shows the results for the `IBLP-DEN` and `IBLP-FIS` sets and Fig. 2b shows the results for the `MIBLP-XU` set. These figures show the superiority of `whenLFixed-LFixed` and `whenXYIntOrLFixed-LFixed` over the other three methods, with roughly the same performance for each. Based on these results, the settings for `whenXYIntOrLFixed-LFixed` have been chosen as the default setting for `MibS` and in the remainder of these experiments unless otherwise noted.

4.2 Impact of branching strategy

As mentioned earlier, the `branchStrategy` parameter controls which variables are considered branching candidates and can be set to `fractional` and `linking`. In order to evaluate the effect of the parameter, we compared the performance of these two methods. The linking solution pool was used in both cases.

In initial testing, we observed a possible relationship between the number of integer first- and second-level variables and the performance of branching strategies. Hence, we further analyzed the results by dividing them into two separate sets with $r_1 \leq r_2$ (27 instances) and $r_1 > r_2$ (98 instances). Figure 3 shows the performance profiles for these two sets with the solution time as the performance measure.

Figure 3a shows that `linking` generally performed better when $r_1 \leq r_2$, while Fig. 3b indicates that the `fractional` branching strategy performed better when $r_1 > r_2$. Based on these results, the default value of `branchStrategy` parameter in `MibS` has been set to `linking` and `fractional` for the instances with $r_1 \leq r_2$ and $r_1 > r_2$, respectively. In general, it would not be easy to predict which branching strategy will behave better for a particular given instance, but it is intuitive that when $r_1 << r_2$, the `linking` strategy would be better.

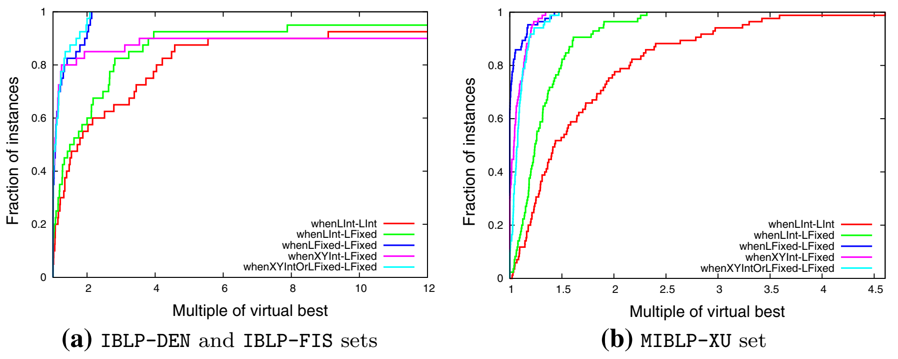


Fig. 2 Impact of the parameters for solving problems (SL-MILP) and (UB)

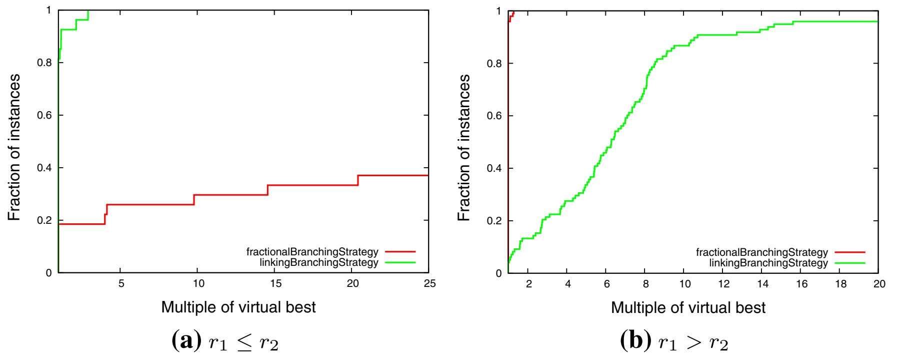
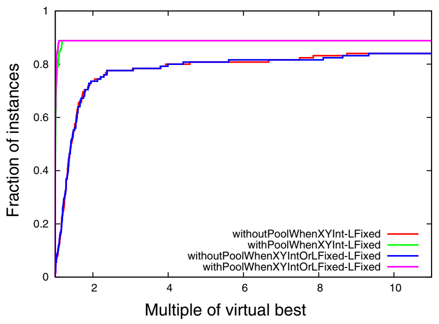
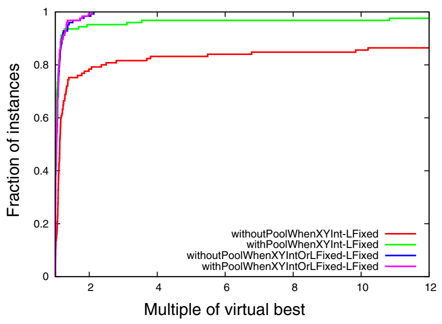


Fig. 3 Impact of the `branchStrategy` parameter



(a) `fractional` branching strategy



(b) `linking` branching strategy

Fig. 4 Impact of the linking solution pool

4.3 Impact of linking solution pool

A set of eight experiments were evaluated to assess the impact of the linking solution pool. The chosen parameters for these experiments were:

- `withoutPoolWhenXYInt-LFixed`: The linking solution pool is not used and `whenXYInt-LFixed` strategy is used for solving problems (SL-MILP) and (UB).
- `withPoolWhenXYInt-LFixed`: The same as the above method, but with the use of the linking solution pool.
- `withoutPoolWhenXYIntOrLFixed-LFixed`: The linking solution pool is not used and `whenXYIntOrLFixed-LFixed` startegy is used for solving problems (SL-MILP) and (UB).
- `withPoolWhenXYIntOrLFixed-LFixed`: The same as the above method, but with the use of the linking solution pool.

Each of the above four settings was tested with both `fractional` and `linking` branching strategies, although the linking pool was only expected to have a large impact when we allow branching on non-linking variables. This is because when branching only on linking variables, linking solutions can only arise again within the same subtree as they first arose and this can only happen if the solution was not already removed with a valid inequality.

The performance profiles shown in Fig. 4 compare the solution time of the described methods. Figure 4a shows the methods which use `fractional` branching strategy, and Fig. 4b shows the methods with `linking` branching strategy.

As expected, when branching was not limited to linking variables, the linking solution pool was effective in decreasing the solution time. This can be observed by comparing

- `withPoolWhenXYInt-LFixed` to
- `withoutPoolWhenXYInt-LFixed`

for both the `fractional` and `linking` branching strategies, as well as

- `withPoolWhenXYIntOrLFixed-LFixed` to

Table 2 Analysis of total time and frequency of solving problems (SL-MILP) and (UB)

Data set	withPoolWhenXYIntOr LFixed-LFixed(Linking)			withoutPoolWhenXYIntOr LFixed-LFixed(Linking)			withPoolWhenXYIntOr LFixed-LFixed(Fractional)			withoutPoolWhenXYIntOr LFixed-LFixed(Fractional)		
	SL count	UB count	Time (%)	SL count	UB count	Time (%)	SL count	UB count	Time (%)	SL count	UB count	Time (%)
IBLP-DEN ($r_1 \leq r_2$)	1,027,234	1,023,867	89	1,029,644	1,023,867	89	203,711	48,190	4	5,570,810	316,227	91
IBLP-DEN ($r_1 > r_2$)	1,300,436	1,184,094	53	1,317,993	1,184,094	55	820,832	2333	17	2,571,199	2355	39
IBLP-FIS ($r_1 \leq r_2$)	699	695	84	699	695	84	201	29	0	1,336,226	31,058	43
IBLP-FIS ($r_1 > r_2$)	91,280	56,014	13	93,013	56,014	13	19,103	0	2	27,980	0	3
MIBLP-XU	26,593	8791	4	26,597	8795	4	5972	5618	18	10,654	10,239	38

– `withoutPoolWhenXYIntOrLFixed-LFixed`

for the `fractional` branching strategy.

In these cases, branching can also be done on non-linking variables. However, use of the solution pool for `withoutPoolWhenXYIntOrLFixed-LFixed` with `linking` branching strategy does not improve performance because the branching was only done on linking variables. Based on the results achieved in this section, the linking solution pool is used by default in `MibS`.

Table 2 shows the total number of instances of (SL-MILP) and (UB) solved when using the methods `withPoolWhenXYIntOrLFixed-LFixed` and `withoutPoolWhenXYIntOrLFixed-LFixed` with both `linking` and `fractional` branching strategies, as well as the percent of total solution time required. Comparing the columns for the `linking` branching strategy verifies the results shown in Fig. 4b. These columns show that using the linking solution pool does not decrease the number of (SL-MILP) and (UB) instances solved, so we do not expect to improve the solution time either. Similarly, the columns for the `fractional` branching strategy verify the results shown in Fig. 4a and show that the number of instances of (SL-MILP) and (UB) can be reduced significantly by using the pool in this case, resulting in decreased solution time.

The results with both the `linking` and `fractional` strategies with the linking pool (the first and third columns) show that fewer instances of problems (SL-MILP) and (UB) are solved in general (and a smaller percentage of time required) with the `fractional` strategy, regardless of whether the overall solution time is smaller with the `linking` strategy or not. This does not mean, however, that using the `fractional` strategy always results in the need to solve fewer problems (SL-MILP) and (UB). It is generally only by taking advantage of the linking solution pool that this is avoided. Comparing `linking` strategy without the pool to the `fractional` strategy without the pool illustrates that the `fractional` strategy sometimes requires solving more problems (SL-MILP). Moreover, Table 2 shows that different data sets do not have the same behavior from the point of required time for solving MILPs.

4.4 Impact of primal heuristics

In order to evaluate the impact of employing primal heuristics implemented in `MibS` (see Sect. 2.5), we compared four different methods:

- `noHeuristics`: Parameters are set to the default values obtained from the results of previous sections, i.e.,
  - the `whenXYIntOrLFixed-LFixed` strategy is used for solving problems (SL-MILP) and (UB).
  - `branchStrategy` is set to `linking` and `fractional` for the instances with $r_1 \leq r_2$ and $r_1 > r_2$, respectively.
  - the linking solution pool is used.
- `impObjectiveCut`: The same as `noHeuristics`, but the improving objective cut heuristic is turned on.

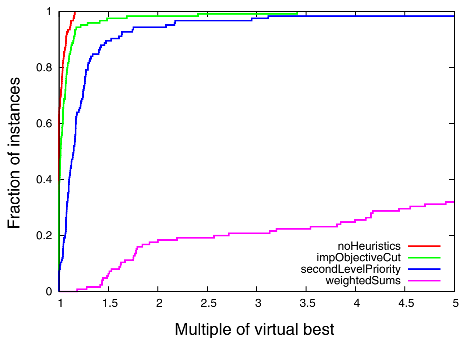


Fig. 5 Impact of the primal heuristics

- `secondLevelPriority`: The same as `noHeuristics`, but the second-level priority heuristic is turned on.
- `weightedSums`: The same as `noHeuristics`, but the weighted sums heuristic is turned on.

Figure 5 shows the performance profiles for these four methods with the solution time as the performance measure. The frequency of using heuristics was set to 100.

Based on Fig. 5, the primal heuristics implemented so far are not effective in improving the solution time. However, it may be possible to improve their performance by parameter tuning. No serious effort has been made so far to do this.

4.5 Impact of MILP solver

SYMPHONY was employed as the MILP solver in all experiments described in previous sections. In order to investigate the impact of the employed MILP solver in solving MIBLPs, we repeated some of these experiments using CPLEX (with its default setting) as the MILP solver. We observed that when the total time for solving the required MILPs was not large and the MILPs were easy to solve, the impact of the change in MILP solver was not considerable. In cases where solution of the MILPs required more effort, CPLEX did reduce the time for solving the MILPs by roughly half on average (but the exact amount of reduction depends highly on the instance).

5 Conclusions

We have presented a generalized branch-and-cut algorithm, which is able to solve a wide range of MIBLPs. The components of this algorithm have been explained in detail and we have discussed precisely how the specific features of MIBLPs can be utilized to solve various classes of problems that arise in practical applications. Moreover, we have introduced `MibS`, which is an open-source solver for MIBLPs. This solver has been implemented based on the branch-and-cut algorithm described in the paper and provides a comprehensive and flexible framework in which a variety of algorithmic options are available. We have demonstrated the performance of `MibS` and shown that it is a robust solver capable of solving generic MIBLPs of modest size. In future papers, we will describe additional details of the methodology in MibS, including the cut generation, which we have not discussed here. Our future plans for MibS include the implementation of additional techniques for generating valid inequalities and the possible addition of alternative algorithms.

Acknowledgements This research was made possible with support from National Science Foundation Grants CMMI-1435453, CMMI-0728011, and ACI-0102687.

References

1. Bard, J.F.: Practical Bilevel Optimization: Algorithms and Applications. Kluwer Academic Publishers (1998)
2. Computational Infrastructure for Operations Research. https://www.coin-or.org (2018)
3. Salmeron, J., Wood, K., Baldick, R.: Analysis of electric grid security under terrorist threat. IEEE Trans. Power Syst. **19**(2), 905–912 (2004)
4. Bienstock, D., Verma, A.: The NK problem in power grids: new models, formulations, and numerical experiments. SIAM J. Optim. **20**(5), 2352–2380 (2010)
5. Zhang, Y., Snyder, L., Ralphs, T., Xue, Z.: The competitive facility location problem under disruption risks. Transp. Res. Part E Logist. Transp. Rev. **93**, 453–473 (2016). https://doi.org/10.1016/j.tre.2016.07.002
6. Gao, J., You, F.: Design and optimization of shale gas energy systems: overview, research challenges, and future directions. Comput. Chem. Eng. **106**, 699–718 (2017)
7. Loridan, P., Morgan, J.: Weak via strong stackelberg problem: new results. J. Glob. Optim. **8**(3), 263–287 (1996)
8. Vicente, L., Savard, G., Júdice, J.: Discrete linear bilevel programming problem. J. Optim. Theory Appl. **89**(3), 597–614 (1996)
9. Israeli, E.: System interdiction and defense. Ph.D. thesis, Naval Postgraduate School (1999)
10. DeNegre, S.: Interdiction and Discrete Bilevel Linear Programming. Ph.D. Lehigh University (2011). http://coral.ie.lehigh.edu/~ted/files/papers/ScottDeNegreDissertation11.pdf
11. Jeroslow, R.: The polynomial hierarchy and a simple model for competitive analysis. Math. Program. **32**, 146–164 (1985)
12. Moore, J.T., Bard, J.F.: The mixed integer linear bilevel programming problem. Oper. Res. **38**(5), 911–921 (1990)
13. Von Stackelberg, H.: Marktform und Gleichgewicht. Julius Springer, Berlin (1934)
14. Bracken, J., McGill, J.T.: Mathematical programs with optimization problems in the constraints. Oper. Res. **21**(1), 37–44 (1973)
15. Wollmer, R.: Removing arcs from a network. Oper. Res. **12**(6), 934–940 (1964)
16. Bard, J.F., Moore, J.T.: An algorithm for the discrete bilevel programming problem. Nav. Res. Logist. **39**(3), 419–435 (1992)
17. Wen, U.P., Huang, A.: A simple tabu search method to solve the mixed-integer linear bilevel programming problem. Eur. J. Oper. Res. **88**, 563–571 (1996)
18. Faísca, N.P., Dua, V., Rustem, B., Saraiva, P.M., Pistikopoulos, E.N.: Parametric global optimisation for bilevel programming. J. Glob. Optim. **38**, 609–623 (2007)
19. DeNegre, S., Ralphs, T.: A branch-and-cut algorithm for bilevel integer programming. In: Proceedings of the Eleventh INFORMS Computing Society Meeting, pp. 65–78 (2009). https://doi.org/10.1007/978-0-387-88843-9_4. http://coral.ie.lehigh.edu/~ted/files/papers/BILEVEL08.pdf
20. Xu, P., Wang, L.: An exact algorithm for the bilevel mixed integer linear programming problem under three simplifying assumptions. Comput. Oper. Res. **41**, 309–318 (2014)
21. Zeng, B., An, Y.: Solving bilevel mixed integer program by reformulations and decomposition. Tech. rep., University of South Florida (2014). http://www.optimization-online.org/DB_FILE/2014/07/4455.pdf
22. Caramia, M., Mari, R.: Enhanced exact algorithms for discrete bilevel linear problems. Optim. Lett. **9**(7), 1447–1468 (2015)

23. Caprara, A., Carvalho, M., Lodi, A., Woeginger, G.J.: Bilevel knapsack with interdiction constraints. INFORMS J. Comput. **28**(2), 319–333 (2016)
24. Hemmati, M., Smith, J.C.: A mixed-integer bilevel programming approach for a competitive prioritized set covering problem. Discrete Optim. **20**, 105–134 (2016)
25. Wang, L., Xu, P.: The watermelon algorithm for the bilevel integer linear programming problem. SIAM J. Optim. **27**(3), 1403–1430 (2017)
26. Lozano, L., Smith, J.C.: A value-function-based exact approach for the bilevel mixed-integer programming problem. Oper. Res. **65**(3), 768–786 (2017)
27. Fischetti, M., Ljubić, I., Monaci, M., Sinnl, M.: On the use of intersection cuts for bilevel optimization. Math. Program. **172**, 77–103 (2018)
28. Fischetti, M., Ljubić, I., Monaci, M., Sinnl, M.: A new general-purpose algorithm for mixed-integer bilevel linear programs. Oper. Res. **65**(6), 1615–1637 (2017)
29. Mitsos, A.: Global solution of nonlinear mixed integer bilevel programs. J. Glob. Optim. **47**, 557–582 (2010)
30. Kleniati, P.M., Adjiman, C.S.: Branch-and-sandwich: a deterministic global optimization algorithm for optimistic bilevel programming problems. Part I: theoretical development. J. Glob. Optim. **60**, 425–458 (2014)
31. Kleniati, P.M., Adjiman, C.S.: Branch-and-sandwich: a deterministic global optimization algorithm for optimistic bilevel programming problems. Part II: convergence analysis and numerical results. J. Glob. Optim. **60**, 459–481 (2014)
32. Padberg, M., Rinaldi, G.: Optimization of a 532-city symmetric traveling salesman problem by branch and cut. Oper. Res. Lett. **6**(1), 1–7 (1987)
33. Padberg, M., Rinaldi, G.: A branch-and-cut algorithm for the resolution of large-scale traveling salesman problems. SIAM Rev. **33**, 60–100 (1991)
34. Land, A.H., Doig, A.G.: An automatic method for solving discrete programming problems. Econometrica **28**, 497–520 (1960)
35. Achterberg, T.: Constraint integer programming. Ph.D. thesis. https://doi.org/10.14279/depositonce-1634 (2007)
36. Achterberg, T., Koch, T., Martin, A.: Branching rules revisited. Oper. Res. Lett. **33**(1), 42–54 (2005)
37. Marchand, H., Martin, A., Weismantel, R., Wolsey, L.: Cutting planes in integer and mixed integer programming. Discrete Appl. Math. **123**(1), 397–446 (2002)
38. Wolter, K.: Implementation of Cutting Plane Separators for Mixed Integer Programs. Master's thesis, Technische Universität Berlin (2006)
39. Balas, E., Jeroslow, R.: Canonical cuts on the unit hypercube. SIAM J. Appl. Math. **23**(1), 61–69 (1972)
40. Balas, E.: Intersection cuts-a new type of cutting planes for integer programming. Oper. Res. **19**(1), 19–39 (1971)
41. Ehrgott, M., Wiecek, M.M.: Multiobjective programming. In: Ehrgott, M., Figueira, J., Greco, S. (eds.) Multiple Criteria Decision Analysis: State of the Art Surveys, pp. 667–722. Springer, Berlin (2005)
42. Geoffrion, A.M.: Proper efficiency and the theory of vector maximization. J. Math. Anal. Appl. **22**, 618–630 (1968)
43. DeNegre, S., Ralphs, T., Tahernejad, S.: version 1.1.2 (2017). https://doi.org/10.5281/zenodo.1439384. https://github.com/coin-or/MibS
44. Xu, Y., Ralphs, T., Ladányi, L., Saltzman, M.: Computational experience with a software framework for parallel integer programming. INFORMS J. Comput. **21**, 383–397 (2009). https://doi.org/10.1287/ijoc.1090.0347
45. Ralphs, T., Ladányi, L., Saltzman, M.: A library hierarchy for implementing scalable parallel search algorithms. J. Supercomput. **28**, 215–234 (2004). https://doi.org/10.1023/B:SUPE.0000020179.55383.ad
46. Xu, Y., Ralphs, T.: ALPS version 1.5.6 (2019). https://doi.org/10.5281/zenodo.245971. https://github.com/coin-or/CHiPPS-ALPS
47. Xu, Y., Ralphs, T., Ladányi, L., Saltzman, M.: ALPS: a framework for implementing parallel tree search algorithms. In: The Proceedings of the Ninth INFORMS Computing Society Conference, vol. 29, pp. 319–334 (2005). https://doi.org/10.1007/0-387-23529-9_21
48. Xu, Y., Ralphs, T.: BiCEPs version 0.94.4 (2017). https://doi.org/10.5281/zenodo.245652. https://github.com/coin-or/CHiPPS-BiCePS

49. Xu, Y., Ralphs, T.: BLIS version 0.94.5 (2017). https://doi.org/10.5281/zenodo.246079. https://github.com/coin-or/CHiPPS-BLIS
50. Forrest, J.J.: Clp version 1. 16 (2017). https://github.com/coin-or/Clp
51. Ralphs, T., Güzelsoy, M., Mahajan, A.: SYMPHONY version 5.6.16 (2017). https://doi.org/10.5281/zenodo.248734
52. Ralphs, T., Güzelsoy, M.: The SYMPHONY callable library for mixed integer programming. In: Proceedings of the Ninth INFORMS Computing Society Conference, pp. 61–76 (2005). https://doi.org/10.1007/0-387-23529-9_5. http://coral.ie.lehigh.edu/~ted/files/papers/SYMPHONY04.pdf
53. Cgl version 0.59 (2017). https://github.com/coin-or/Cgl
54. Osi version 0.107 (2017). https://github.com/coin-or/Osi
55. Hassanzadeh, A., Ralphs, T.: A Generalized Benders' Algorithm for Two-Stage Stochastic Program with Mixed Integer Recourse. Tech. rep., COR@L Laboratory Report 14T-005, Lehigh University (2014). http://coral.ie.lehigh.edu/~ted/files/papers/SMILPGenBenders14.pdf
56. Ralphs, T., Güzelsoy, M.: Duality and warm starting in integer programming. In: The Proceedings of the 2006 NSF Design, Service, and Manufacturing Grantees and Research Conference (2006). http://coral.ie.lehigh.edu/~ted/files/papers/DMII06.pdf
57. Ibm ILOG CPLEX Optimizer. https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/ (2017)
58. Forrest, J.J.: Cbc version 2.9 (2017). https://projects.coin-or.org/Cbc
59. Bixby, R.E., Ceria, S., McZeal, C.M., Savelsbergh, M.W.: An updated mixed integer programming library: Miplib 3.0. Tech. rep. (1998)