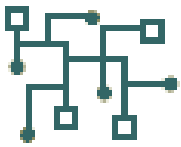


ISE

Industrial and Systems Engineering

# A Branch-and-Cut Algorithm for Mixed Integer Bilevel Linear Optimization Problems and Its Implementation


Sahar Tahernejad and Ted K. Ralphs

Department of Industrial and System Engineering, Lehigh University, Bethlehem, PA

Scott T. DeNegre

Hospital for Special Surgery, New York, New York



COR@L Technical Report 16T-015-R3


LEHIGH UNIVERSITY

COR@L

COMPUTATIONAL OPTIMIZATION RESEARCH AT LEHIGH

# A Branch-and-Cut Algorithm for Mixed Integer Bilevel Linear Optimization Problems and Its Implementation

Sahar Tahernejad[*1], Ted K. Ralphs[†1], and Scott T. DeNegre[‡2]

[1]Department of Industrial and System Engineering, Lehigh University, Bethlehem, PA
[2]Hospital for Special Surgery, New York, New York

Original Publication: Dec 30, 2016
Last Revised: April 25, 2017

**Abstract**

In this paper, we describe a comprehensive algorithmic framework for solving mixed integer bilevel linear optimization problems (MIBLPs) by 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.

## 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 applications in which multiple, possibly competing decision-makers (DMs), make a sequence of decisions over time (defined formally in Section 1.1). For an ever-increasing array of such applications, the traditional framework of mathematical optimization, which assumes a single *decision maker* (DM) with a single objective function making a decision at a single point in time, is inadequate for analysis. The motivation for the development of `MibS`, which was begun a decade ago when little was known about solution of these problems, is both to serve as an open test bed for new algorithmic ideas and simply to provide

[*]sat214@lehigh.edu
[†]ted@lehigh.edu
[‡]DeNegreS@hss.edu

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*. In a multilevel optimization problem, 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 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 DMs 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 second-level decisions, which in turn impact the objective value of the first-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 effect 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 [Bard, 2013].

Thanks to the steady improvement in solution methodologies for traditional optimization problems and the availability of open source solvers [COIN-OR], the development of practical solution methods for MIBLPs has become more realistic. This possibility 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. 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 simplest 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 $K \subseteq \{1, \dots, n_1\}$ as $x_K$ (and similarly for sub-vectors of $y \in \mathbb{R}^{n_2}$).

The general form of an MIBLP is then

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

where the function $\Xi$ 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 in which

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

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 $A^1 \in \mathbb{Q}^{m_1 \times n_1}$, $G^1 \in \mathbb{Q}^{m_1 \times n_2}$, $b^1 \in \mathbb{Q}^{m_1}$, $A^2 \in \mathbb{Q}^{m_2 \times n_1}$, $G^2 \in \mathbb{Q}^{m_2 \times n_2}$.

Note that it is customary in the literature on bilevel optimization for the parameterized 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. For consistency with results in other papers, we use the more general form here.

As is customary, we define $\Xi(x) = \infty$ in the case of infeasibility of problem on the right-hand side of (RF). The pessimistic variant is obtained simply by considering

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

Other variants are, of course, also possible. The algorithms discussed herein can easily be adapted to these variants.

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 lead 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 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 presented in (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 "project out" the second-level variables. In contrast with the Benders method for traditional optimization problems, however, the second-level variables are re-introduced in solving the relaxation we employ in our branch-and-cut algorithm (see Section 2.1). Second, it is necessary at certain points in the algorithm to evaluate $\Xi(x)$ explicitly, and it is convenient to give it an explicit form here to make this step clearer. Evaluating $\Xi(x)$ is itself a bilevel optimization problem, but a much easier one to solve in general than the original problem.

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

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

where $\phi$ 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

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

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 $\phi(A^2 x)$ to be rendered a constant and for (MIBLP-VF) to reduce to an MILP. This result is stated formally in Section 2.2. Because of their central importance in what follows, we now formally define the concept of a *linking variable*.

**Definition 1** (Linking Variables)**.** *Let*

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

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

Per our earlier cited 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 [Vicente et al., 1996].

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

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 $x \in X$, 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 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 implies that $G^2 r < 0$ for all $r \in \mathbb{R}^{n_2}_+$).

**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 our ultimate goal of solving (MIBLP) is to determine $x \in X$ minimizing $cx + \Xi(x)$, this can also be interpreted as 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 $x \in \mathbb{R}^{n_1}$, 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 Figure 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 Section 1.3.

## 1.2 Special Cases

There are a number of special 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

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

where

$$\mathcal{P}_1^{INT} = \left\{ x \in \mathbb{R}^n_+ \;\middle|\; A^1 x \geq b^1 \right\},$$
$$\mathcal{P}_2^{INT}(x) = \left\{ y \in \mathbb{R}^n_+ \;\middle|\; G^2 y \geq b^2, y \leq \operatorname{diag}(u)(e - x) \right\},$$

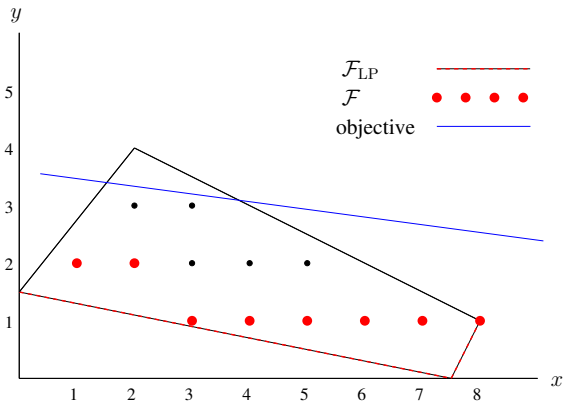
$e$ represent an n-dimensional vector of ones, $u_i \in \mathbb{R}$ denotes the upper-bound of $y_i$ for $i = 1, ..., n$.

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, even for the case in which all variables are continuous ($r_1 = r_2 = 0$), the problem is NP-hard. The general case described is in the class $\Sigma_p^2$-hard, 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 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-stage 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 bilevel linear optimization problem is in $X \times Y$, it is not necessarily optimal for (MIBLP). These properties can be seen in Figure 1, which displays a well-known example originally from Moore and Bard [1990] that continues to be used by tradition for illustrating these concepts.




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

Figure 1: The feasible region of IBLP [Moore and Bard, 1990].

### 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 seminal work on which was done by Von Stackelberg [1934]. The first bilevel optimization formulations in the form presented here were introduced and the term was coined in the 1970s by [Bracken and McGill, 1973], but computational aspects of related optimization problems have been studied since at least the 1960s (see, e.g., Wollmer [1964]). Study of algorithms for the case in which integer variables appear is generally acknowledged to have been launched by Moore and Bard [1990], 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 [1992], Wen and Huang [1996] and Faísca et al. [2007] studied the bilevel problems with binary variables. Bard and Moore [1992] proposed an exact algorithm for *integer bilevel optimization problems* (IBLPs) in which all variables are binary. Wen and Huang [1996], 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. [2007] 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 [2009] and DeNegre [2011] generalized the existing framework of branch-and-cut that is the standard approach for solving MILPs to the case of IBLPs, introducing the *integer no-good cut* to separate bilevel infeasible solution from the convex hull of bilevel feasible solutions (see Section 2.4). Xu [2012] proposed the *watermelon algorithm* for solving IBLPs, which removes bilevel infeasible solutions that satisfy integrality constraints by a non-binary branching disjunction. Xu and Wang [2014] 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 [2014] for solving general MIBLPs. Their method finds the optimal solution if it is attainable, otherwise it finds an $\epsilon$-optimal solution. Caramia and Mari [2015] 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 [DeNegre and Ralphs, 2009]. Knapsack interdiction problems were studied by Caprara et al. [2016] and they proposed an exact algorithm for these problems. Hemmati and Smith [2016] formulated competitive prioritized set covering problem as an MIBLP and proposed a cutting plane algorithm for solving it. Fischetti et al. [2016a] 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 Fischetti et al. [2016b], they extended their algorithm by suggesting new types of intersection cuts for IBLPs and introduced the so-called *hypercube intersection cut*, valid for MIBLPs in which the linking variables are discrete. Finally, Mitsos [2010] and Kleniati and Adjiman [2014a,b] considered the more general case of mixed integer bilevel non-linear optimization.

### 1.5 Overview of Branch and Cut

Branch and cut is a variant of the well-known branch-and-bound algorithm, 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 branch and cut 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 subproblems 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 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 relaxation that can be proven not to contain any *improving* feasible solutions 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 rather 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 a valid *improving* disjunction in order to distinguish it from the more standard definition, refers to a collection of sets that contain no

*improving* feasible solution. In the remainder of the paper, we drop the word "improving" when referring to such disjunctions.

**Definition 2** (Valid Improving Disjunction)**.** *A* valid (improving) 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*

$$\{x \in \mathcal{X} \mid f(x) < f(x^*)\} \subseteq \bigcup_{1 \leq i \leq k} X_i \tag{1}$$

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 Section 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 $t \in T$ 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 are aggregated to obtain the global upper bound

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

which represents the best solution found so far (known as the *incumbent*), while lower bounds are similarly aggregated to form the global lower bound

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

These global bounds are updated as the algorithm progresses and when the global lower and upper bounds are 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 $L^t = U^t$, 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 Sections 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 resulting subproblems are optimization problems of the form

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

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

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

is always valid when $(\pi, \pi_0) \in \mathbb{Z}^{n+1}$ and $\pi_i = 0$ for $i > r$, since we must 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 $\pi$ is the $i^{\text{th}}$ 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 Section 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 of the lower bound). In practice, one usually employs hybrids that balance these two goals.

**Cutting.** The *branch-and-cut algorithm* is an extension to the basic strategy of branch and bound that can improve the performance of branch-and-bound approach by strengthening the relaxation with *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** (Valid Improving Inequality)**.** *The pair* $(\alpha, \beta) \in \mathbb{R}^{n+1}$ *is a* valid improving inequality *for* (GO) *with respect to* $x^* \in \mathcal{X}$ *if*

$$\{x \in \mathcal{X} \mid f(x) < f(x^*)\} \subseteq \{x \in \mathbb{R}^n \mid \alpha x \geq \beta\}$$

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.

A common method for generating valid inequalities in the case of MILP which generalizes naturally to our setting is that of generating *disjunctive inequalities*, which are derived relative to some relaxed feasible set $\mathcal{Q} \supseteq \mathcal{X}$ (usually a convex set that is also utilized in formulating the lower bounding problem) and a given valid disjunction.

**Definition 4** (Disjunctive Inequality)**.** *A* disjunctive inequality *with respect to* $\mathcal{Q} \supseteq \mathcal{X}$ *and a disjunction* $\{X_i\}_{1 \leq i \leq k}$ *valid for* $\mathcal{X}$ *is any inequality valid for*

$$\mathcal{Q} \bigcap \left( \bigcup_{1 \leq i \leq k} X_i \right),$$

A subclass of the disjunctive inequalities are the *split inequalities*, of which the *Chvátal* inequalities are a further subclass.

**Definition 5** (Split Inequality)**.** *A* split inequality *with respect to* $\mathcal{Q} \supseteq \mathcal{X}$ *and a disjunction* $X_1$ *and* $X_2$ *of the form* (GD) *valid for* $\mathcal{X}$ *is any inequality valid for* $\mathcal{Q} \cap (X_1 \cup X_2)$*.*

A further subclass of split inequalities are the *Chvátal inequalities.*

**Definition 6** (Chvátal Inequality)**.** *A* Chvátal inequality *is a split inequality for which* $\mathcal{X} \cap X_2$ *does not contain any improving solutions (and hence,* $(\pi, \pi_0)$ *in* (GD) *is itself a valid inequality).*

In Section 2.4, we show how these concepts can be applied in deriving inequalities valid for $\mathcal{F}$. Finally, we also derive inequalities from a related class, the intersection cuts, for whose description we refer the reader to [Balas, 1971]. For an excellent treatment of the relationship between all of these classes in the case of MILP, see Conforti et al. [2014].

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 Section 2.4.

### 1.6 Outline

The remainder of the paper is organized as follows. In Section 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 Section 4, we discuss computational results. Finally, in Section 5, we conclude the paper with some final thoughts.

## 2 A Branch-and-cut Algorithm

The algorithm implemented in `MibS` is based on the basic algorithmic framework originally described by DeNegre and Ralphs [2009], 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 launching into specific discussion of various components.

As we have described earlier in Section 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 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 (corresponding to the regions removed by imposing variables disjunctions or valid inequalities) or by imposing an *objective cut* that removes all non-improving solutions. 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 $\phi$ and the risk function $\Xi$. In both cases, it's 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 both

$$\begin{aligned} \Xi(x^1) &= \Xi(x^2) \text{ and} \\ \phi(A^2 x^1) &= \phi(A^2 x^2) \end{aligned}$$

whenever $x^1_L = x^2_L$. 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 Section 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 of these 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 valid inequalities described in Section 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

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

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'll describe shortly employs only changes to the bounds of variables, the subproblems that arise have feasible regions of the form

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

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. $\mathcal{F}^t$ is thus itself the feasible region of the MIBLP

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

so we can apply a similar relaxation to obtain the bound

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

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 Section 2.2), then node $t$ should be discarded.

Otherwise, the next step is to check feasibility of

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

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 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 Section 3. We may thus defer it until later, depending on the values of parameters described in Section 3.1.

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

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

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 Section 3.3). 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 $i \notin L$), since this may still lead either to the discovery of a new bilevel feasible solution or a proof that $x^t \notin \mathcal{F}_1$. We discuss algorithmic strategies for specifically when we must or can optionally solve (SL-MILP) in Section 3. 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 $\hat{y}^t \in \mathcal{P}_1(x^t)$. This condition is satisfied vacuously in the usual case of $G^1 = 0$. If $(x^t, \hat{y}^t)$ is feasible, then we can set $U^t = cx^t + d^1 \hat{y}^t$ and update the global upper bound if $U^t < U$, as before.

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

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

so 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 Fischetti et al. [2016a] and DeNegre [2011]) and is also the basis for an algorithm by Mitsos [2010]. 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 (1) it may improve the global upper bound, (2) 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 (3) 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 Section 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 it is important to observe that even though $(x^t, \hat{y}^t)$ is bilevel feasible, we may still have $d^1\hat{y}^t > \Xi(x^t)$ implying that there exists a rational reaction $\bar{y} \in \mathcal{R}(x^t) \cap \mathcal{P}_1(x^t)$ to $x^t$ with $d^1\bar{y} < d^1\hat{y}^t$. 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 under taken in others, depending on the parameter settings described in detail in Section 3.

To calculate $\Xi(x^t)$, we observe that if the values of the first-level variables are fixed, then the problem of calculating the optimal values of the second-level variables is equivalent to

$$\Xi(x^t) = \min \left\{ d^1 y \;\middle|\; y \in Y \cap \mathcal{P}_1(x^t) \cap \mathcal{P}_2(x^t), d^2 y \leq \phi(b^2 - A^2 x^t) \right\}, \tag{3}$$

which is an MILP since we have already computed $\phi(b^2 - A^2 x^t)$ 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 (3) is still an MILP even when the values of first-level variables not in set $L$ are unfixed. This means that solving a variation of (3) 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 $L^t \geq U$. Second, when the relaxation is infeasible, we have $L^t \geq \infty$ and we can again prune the node.

There is one additional case that is particular to MIBLPs and utilizes the property of MIBLPs illustrated in the next theorem.

**Theorem 1** ([Fischetti et al., 2016a]). *For $\gamma \in \mathbb{Z}^L$, we have*

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

*Proof.* From the definitions of sets $\mathcal{S}$ and $\mathcal{F}$ provided in Section 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*

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

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-stage 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*

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

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) provides the bilevel feasible solution which is at least as good as the optimal solution of node $t$. Hence, node $t$ can be pruned after solving either ($UB^t$) or (UB). Note that these problems may be infeasible, which shows that node $t$ is infeasible. Although solving problem ($UB^t$) may be easier comparing with (UB), solving problem (UB) provides useful information about all bilevel feasible solutions with $x_L = x^t_L$ for the whole tree (not only node $t$), so we recommend solving problem (UB) instead of ($UB^t$) (see Sections 3.1 and 3.2).

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 maximize potential improvement in the upper bound. We discuss algorithmic strategies for specifically when we must or can optionally solve (UB) in Section 3.

### 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 so that it also removes the solution to the current relaxation.

When the branching procedure is invoked for node $t$, we have that either

- the solution $(x^t, y^t) \notin \mathcal{F}$ because
  - $(x^t, y^t) \notin X \times Y$,
  - $d^2y^t > \phi(A^2x^t)$ (we have previously solved (SL-MILP) with $\gamma = x^t_L$).

  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 Section 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 $t$ ($(l_x^t)_L = (u_x^t)_L$), the node can be pruned after solving (UB) (which is an MILP). 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 while any such variables remain unfixed. We show in Section 4 that the apparent logic in such a scheme does also translate into empirical effectiveness in some cases, but such a 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 ($x_L^t \in \mathbb{Z}^L$) and yet we would still like to (or need to) impose a branching disjunction.

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

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

as usual. When $x_L^t \in \mathbb{Z}^L$, however, it is less obvious what to do if we still wish to restrict to branching only on linking variables. If we either have $x_i^t \notin \mathbb{Z}$ for $i \notin L$ or we have that $y \notin Y$, 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 is not a good idea empirically (see Section 4) and this is in concert with our intuition.

Unfortunately, when $x_L^t \in \mathbb{Z}^L$, 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 $x_i^t \in \mathbb{Z}$ (and assuming that $l_{x_i}^t \neq u_{x_i}^t$), we may branch, e.g., either on the valid disjunction

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

or

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

preferring the former when $x_i^t \leq u_i^t - 1$ (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 $\mathcal{S}^t = \mathcal{P}^t \cap (X \times Y)$, we have that $x^t$ will be contained in the projection of the feasible region of exactly one of these leaf nodes. In fact, for the leaf node whose feasible region contains $x^t$, we must have that the values of all integer linking variables are fixed, since we would otherwise continue

branching. The projection of the union of the feasible regions of the remaining leaf nodes is thus equal to

$$\text{proj}_x(\mathcal{S}_t) \setminus \left\{x \in X \mid x_L = x_L^t\right\}$$

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

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

For the leaf node whose feasible region contains $x^t$, all linking variables have been fixed, so we can no longer branch according to the described scheme. We have already noted, however, that we can prune such a node after solving problem (UB).

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 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 Section 3.1).

- We may prune the node after solving problem (UB), as explained in Section 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 Section 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., [Achterberg et al., 2005]). 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 among 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 non-improving solutions. We refer the reader to Marchand et al. [2002] and Wolter [2006] for theoretical background on the separation problem and cutting plane methods in general. In the case of MIBLP, the cutting plane method is to iteratively separate the solution to the current relaxation from $\text{conv}(\{(x,y) \in \mathcal{F} \mid cx + d^1y < U\})$, the convex hull of improving solutions. DeNegre and Ralphs [2009] showed that 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 considered, we apply here the same notion of "valid inequality" that we introduced in Definition 3, but re-state the definition here 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 a *valid inequality* for $\mathcal{F}$ if

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

where $U$ is the current global upper bound. The addition of such valid inequalities has the benefit both of removing irrelevant solutions and of strengthening the relaxation, thereby improving the lower bound.

In Section (2.1), we explained that the difference between the feasible region of a subproblem and that of the original problem is only in changes to the bounds of variables, but when generating valid inequalities, the constraints of the relaxation may also be modified. The feasible region $\mathcal{P}^t$ of the relaxation at node $t$ from ($\text{LR}^t$) is therefore updated to

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

where $\Pi^t$ is a polyhedron representing the valid inequalities applied at node $t$. These may include inequalities generated at any of the ancestor nodes in the path to the root node in the search tree.

There are several distinct categories of valid inequalities that can be generated, depending on the feasibility conditions violated by the solution $(x^t, y^t)$ to the relaxation we are trying to remove at node $t$. Generally speaking, we may generate inequalities valid for any of

- $\text{conv}\left(\left\{(x,y) \in \mathcal{S} \;\middle|\; cx + d^1y < U\right\}\right)$ ("feasibility" cuts)
- $\text{conv}\left(\left\{(x,y) \in \mathcal{F} \;\middle|\; cx + d^1y < U\right\}\right)$ ("optimality" cuts)
- $\text{conv}\left(\{(x \in \mathcal{F}_1 \mid cx + \Xi(x) < U\}\right)$ ("projected optimality" cuts)

The feasibility cuts include those classes generally employed in solution of MILPs, but also include other types that we discuss later in this section. Inequalities used to remove solutions $(x^t, y^t) \in \mathcal{S}$ at node $t$ may be referred to roughly as "optimality cuts", although these three classes are not entirely distinct and the categorization is meant only to provide a rough idea. For example, inequalities that remove one or more specific (possibly feasible) solutions that no longer need to be considered (such as solutions with the same linking component as $x^t$ once $\Xi(x^t)$ has been computed) could technically be considered as being either feasibility or optimality cuts, depending on one's point of view.

`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 will be provided in a future paper that analyzes computational results in more detail. Since each class of valid inequalities presented here is valid for different problems types, we say specifically what assumptions have to be satisfied for validity in each case.

### 2.4.1 Generalized No-good Cut

**Validity:** Valid for problems in which all linking variables are binary.

**Discussion:** This cut is a Chvátal inequality that generalizes the *no-good cut* introduced by DeNegre [2011] in the context of MIBLPs. Similar cuts have been used in many other contexts and were probably first suggested by Balas and Jeroslow [1972]. The purpose of this cut is to remove all solutions for which the linking variables have certain fixed values.

**Theorem 2.** *Suppose all linking variables are binary. Then, for $\gamma \in \mathbb{B}^L$ and $(x, y) \in \mathcal{F}$, we have*

$$\sum_{i \in L:\gamma_i=0} x_i + \sum_{i \in L:\gamma_i=1} (1 - x_i) \geq 1 \Leftrightarrow x_L \neq \gamma. \tag{4}$$

*Proof.* When $x_L = \gamma$, we have $\sum_{i \in L:\gamma_i=0} x_i = 0$ and $\sum_{i \in L:\gamma_i=1}(1 - x_i) = 0$, which follows that

$$x_L = \gamma \Rightarrow \sum_{i \in L:\gamma_i=0} x_i + \sum_{i \in L:\gamma_i=1} (1 - x_i) = 0. \tag{5}$$

On the other hand, when $x_L \neq \gamma$, at least one of the following happens

- $\exists i \in L$ so that $\gamma_i = 0$ and $x_i = 1 \Rightarrow \sum_{i \in L:\gamma_i=0} x_i \geq 1$.
- $\exists i \in L$ so that $\gamma_i = 1$ and $x_i = 0 \Rightarrow \sum_{i \in L:\gamma_i=1}(1 - x_i) \geq 1$.

Hence, we have

$$x_L \neq \gamma \Rightarrow \sum_{i \in L:\gamma_i=0} x_i + \sum_{i \in L:\gamma_i=1} (1 - x_i) \geq 1. \tag{6}$$

The result follows from (5) and (6). □

**Application**: If at node $t$, we have $x_L^t \in \mathbb{Z}^L$, if either we choose to solve (SL-MILP) and it is infeasible or we choose to solve (UB), then we can store $x_L^t$ in the linking solution pool (if applicable) and add a valid inequality of the form (4) 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.

### 2.4.2 Integer No-Good Cut

**Validity:** Valid for problems in which $X = \mathbb{Z}^{n_1}$, $Y = \mathbb{Z}^{n_2}$ and all problem data are integer (with the possible exception of the objective function).

**Discussion:** This cut, introduced by DeNegre [2011], is also a Chvátal inequality in which the disjunction is derived by taking combinations of inequalities in the description of $\mathcal{P}$ in a fashion like that used in deriving similar cuts valid for the feasible regions of MILPs.

**Theorem 3** ([DeNegre and Ralphs, 2009]). *Suppose $X = \mathbb{Z}^{n_1}$ and $Y = \mathbb{Z}^{n_2}$ and all constraint coefficients ($A^1, A^2, G^1$ and $G^2$) and right-hand side vector $b^1$ are integer. Moreover, let $(x^*, y^*) \in \mathcal{S} \setminus \mathcal{F}$ be a basic optimal solution to (LR) and $H_1$ and $H_2$ denote the set of indices of first-level and second-level constraints, respectively, binding at $(x^*, y^*)$. Then we have*

$$\alpha^x x + \alpha^y y \geq \beta \quad \forall (x, y) \in \mathcal{F},$$

*where*

$$\alpha^x = \sum_{i \in H_1} a_i^1 - \sum_{i \in H_2} a_i^2, \alpha^y = \sum_{i \in H_1} g_i^1 + \sum_{i \in H_2} g_i^2, \beta = \sum_{i \in H_1} b_i^1 + 1$$

*and $a_i^1, a_i^2, g_i^1, g_i^2$, and $b_i^1$ represent the $i^{th}$ rows of $A^1, A^2, G^1, G^2$ and $b^1$, respectively. Furthermore, we have*

$$\alpha^x x^* + \alpha^y y^* = \beta - 1.$$

*so the inequality is violated by $(x^*, y^*)$.*

*Proof.* Let $(\alpha^x, \alpha^y, \beta) \in \mathbb{Z}^{n_1 \times n_2 + 1}$ be such that

1. $\{(x, y) \in \mathcal{P} \mid \alpha^x x + \alpha^y y = \beta - 1\} = \{(x^*, y^*)\}$ and
2. $\{(x, y) \in \mathbb{R}^{n_1 \times n_2} \mid \alpha^x x + \alpha^y y \geq \beta - 1\} \supseteq \mathcal{P}$.

We have that $\alpha^x x + \alpha^y y \in \mathbb{Z}$ for all $(x, y) \in X \times Y$. Furthermore,

$$\alpha^x x + \alpha^y y > \beta - 1 \; \forall (x, y) \in (\mathcal{P} \setminus \{(x^*, y^*)\}) \supseteq \mathcal{F}.$$

It follows that $\alpha^x x + \alpha^y y \geq \beta$ for all $(x, y) \in \mathcal{F}$ and that $\alpha^x x^* + \alpha^y y^* < \beta$. Then we need only observe that $(\alpha^x, \alpha^y, \beta)$ as chosen in the theorem satisfy the conditions 1 and 2 above. □

Note that there are variations on this theme that would not make such stringent assumptions, but could make the cut more difficult to compute/verify. In fact, any $(\alpha^x, \alpha^y, \beta)$ satisfying conditions 1 and 2 in the proof of Theorem 3 will constitute an inequality valid for $\text{conv}(\mathcal{F})$ and violated by $(x^*, y^*)$. Theorem 3 provides a convenient way to generate such an inequality under the given assumptions, but other methods for generating such an inequality might enable relaxation of these assumptions.

**Application:** This cut is similar in philosophy to the no-good cut of Theorem (2), but it instead eliminates a single bilevel infeasible solution from the feasible region.

### 2.4.3 Intersection Cut

**Validity:** Valid for problems in which $G^2y - A^2x \in \mathbb{Z}$ for all $(x, y) \in \mathcal{F}$.

**Discussion:** Fischetti et al. [2016a] suggested this cut, which is based on the value function bound (2) and the concept of an intersection cut [Balas, 1971].

**Theorem 4** ([Fischetti et al., 2016a])**.** *Suppose $G^2y - A^2x \in \mathbb{Z}$ for all $(x, y) \in \mathcal{F}$ and let $(x^*, y^*)$ be a basic feasible solution to (LR) such that $(x^*, y^*) \notin \mathcal{F}$ and $\hat{y} \in \mathcal{R}(x^*)$. Then if $(\alpha^x, \alpha^y, \beta) \in \mathbb{R}^{n_1+n_2+1}$ is the intersection cut with respect to the convex set*

$$\mathcal{C} = \left\{(x, y) \in \mathbb{R}^{n_1 \times n_2} \;\middle|\; d^2y \geq d^2\hat{y}, G^2\hat{y} \geq A^2x - 1\right\},$$

*and $(x^*, y^*)$, then*

$$\alpha^x x + \alpha^y y \geq \beta \quad \forall (x, y) \in \mathcal{F}. \tag{7}$$

*Furthermore, $\alpha^x x^* + \alpha^y y^* < \beta$.*

*Proof.* The set $\mathcal{C}$ is a convex set that contains no bilevel feasible solutions in its interior. The validity of the inequality follows from the validity of the procedure for generating intersection cuts given in [Balas, 1971]. An intersection cut is guaranteed to be violated by the basic feasible solution with respect to which it was generated, provided such solution is in the interior of the convex set. We have that $(x^*, y^*)$ is in the interior of $\mathcal{C}$, so the second part of the result follows. □

**Application:** The infeasible solution $(x^t, y^t)$ can be removed from the feasible region by generating the intersection cut (7) with $\hat{y} \in \mathcal{R}(x^t)$.

### 2.4.4 Hypercube Intersection Cut

**Validity:** Valid for problems satisfying Assumption 1.

**Discussion:** Fischetti et al. [2016b] suggested this cut, which is also based on the concept of an intersection cut.

**Theorem 5** ([Fischetti et al., 2016b])**.** *Suppose Assumption 1 is satisfied. Let $(x^*, y^*)$ be a basic feasible solution to (LR). Then if $(\alpha^x, \alpha^y, \beta) \in \mathbb{R}^{n_1+n_2+1}$ is the intersection cut with respect to the convex set*

$$\mathcal{C} = \left\{(x, y) \in \mathbb{R}^{n_1 \times n_2} \;\middle|\; x_i^* - 1 \leq x_i \leq x_i^* + 1 \quad \forall i \in L\right\}$$

*and* $(x^*, y^*)$*, then*

$$\alpha^x x + \alpha^y y \geq \beta \quad \forall (x, y) \in \mathcal{F} \text{ such that } x_L \neq x_L^*. \tag{8}$$

*Furthermore,* $\alpha^x x^* + \alpha^y y^* < \beta$*.*

*Proof.* The set $\mathcal{C}$ is a convex set that contains bilevel feasible solutions $(x, y)$ satisfying $x_L = x_L^*$ in its interior. The result follows from the validity of the procedure for generating intersection cuts given in [Balas, 1971]. An intersection cut is guaranteed to be violated by the basic feasible solution with respect to which it was generated, provided such solution is in the interior of the convex set. We have that $(x^*, y^*)$ is in the interior of $\mathcal{C}$, so the second part of the result follows. □

**Application:** If at node $t$, we have $x_L^t \in \mathbb{Z}^L$, we solve (UB) and store the solution in the linking solution pool (if applicable). After doing so, we can add a valid inequality of the form (8). 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.

### 2.4.5 Increasing Objective Cut

**Validity:** Valid for problems in which linking variables are binary and $A^2 \geq 0$.

**Discussion:** DeNegre [2011] proposed this cut, which is a disjunctive cut derived from the following valid disjunction and is based on the value function bound (2).

$$X_1 = \left\{ (x, y) \in \mathbb{R}^{n_1 \times n_2} \;\middle|\; \sum_{i: x_i^t = 0} x_i = 0, d^2 y \leq d^2 \hat{y} \right\} \qquad X_2 = \left\{ (x, y) \in \mathbb{R}^{n_1 \times n_2} \;\middle|\; \sum_{i: x_i^t = 0} x_i \geq 1 \right\},$$

where $\hat{y} \in \mathcal{R}(x^t)$. The specific way in which this disjunction was exploited by [DeNegre, 2011] is captured in the following theorem, but there are other inequalities that could also be derived from this disjunction.

**Theorem 6** ([DeNegre, 2011])**.** *Suppose all linking variables are binary and* $A^2 \geq 0$*. Then if* $(x^*, y^*) \notin \mathcal{F}$ *and* $\hat{y} \in \mathcal{R}(x^*)$*, we have*

$$d^2 y \leq d^2 \hat{y} + M \left( \sum_{i \in L: x_i^* = 0} x_i^* \right) \quad \forall (x, y) \in \mathcal{F} \tag{9}$$

*where* $M \geq \max\{d^2 y \mid (x, y) \in \mathcal{F}\} - d^2 \hat{y}$*. Furthermore, the inequality* (9) *is violated by* $(x^*, y^*)$*.*

**Application:** This cut can be applied to eliminate $(x^t, y^t)$ from the feasible region at node $t$ when $(x^t, y^t)$ is found to be infeasible. If $\hat{y} \in \mathcal{R}(x^t)$, then a cut of the form (9) will be violated by $(x^t, y^t)$. Note that by solving (UB) following the feasibility check, we could also apply the stronger no-good cut in this situation.

### 2.4.6 Benders Cut

**Validity.** Valid for interdiction problem (MIPINT) with $G^2 \leq 0$.

**Discussion:** This cut was originally mentioned by Caprara et al. [2016] in the context of knapsack interdiction problems, but is valid for a broader class of problems. It utilizes the value function bound (2).

**Theorem 7.** *Let $\mathcal{F}$ be the feasible set of an MIBLP of the form (MIPINT) with $G^2 \leq 0$. Then if $(x^*, y^*) \in \mathcal{F}$, we have*

$$d^2 y \leq \sum_{i=1}^{n_1} d_i^2 y_i^* (1 - x_i) \quad \forall (x, y) \in \mathcal{F} \tag{10}$$

**Application:** A new such cut can be imposed anytime we find a new feasible solution. If $(x^t, y^t)$ is a bilevel infeasible solution at node $t$ and $y^* \in \mathcal{R}(x^t)$ is a bilevel feasible solution found during the feasibility check, then a cut of the form (10) will be violated by $(x^t, y^t)$.

### 2.4.7 Bound Cut

**Validity:** The bound cut is a general cut that is valid for all bilevel optimization problems.

**Discussion:** It is possible to remove a part of $\mathcal{P}$ that does not include any bilevel feasible solutions by generating a strong upper-bound for the second-level problem and adding the following constraint, a so-called *bound cut* to the set of constraints of first-level problem.

$$d^2 y \leq \eta,$$

where $\eta$ is an upper bound for the second-level problem. This cut is valid as long as we have

$$\eta \geq \max_{(x,y) \in \mathcal{F}} \quad d^2 y. \tag{11}$$

Note that the problem of finding the smallest possible value of $\eta$ is identical to problem (MIBLP) except for the objective function.

**Application:** Although the problem on the right-hand side of (11) has the same feasible region as (MIBLP), it is a zero-sum problem, whether the original problem is or not. This may make this bound much easier to compute in some cases. Note also that we don't need to find the smallest possible value of $\eta$ for validity. The required time for computing $\eta$ can be decreased by sacrificing its quality. If we attempt to solve the problem on the right-hand-side of (11) by a branch-and-cut algorithm, we can terminate at any time and still obtain a valid upper bound (note we are maximizing). There are a number of different criteria that can be used for early termination of the solution of problem (11). These include

- Solution time limit
- Duality gap limit

- Number of processed nodes limit

Since there is a trade off between the required time for generating $\eta$ and the strength of the cut, selecting an effective termination criterion is crucial.

Note also that bound cuts can be generated at any node in the search tree. Local validity of a bound cut only requires generating a valid bound with respect to the subproblem under consideration. At nodes deep in the search tree, this may make this problem tractable. Furthermore, the frequency of generating the bound cut affects the total effort and this needs to be carefully tuned as well.

## 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. Four different heuristics introduced in DeNegre [2011] are implemented in `MibS`, as follows.

### 2.5.1 Improving Objective Cut Heuristic

Let $(\bar{x}, \bar{y}) \in \mathcal{S}$ and $\hat{y} \in \mathcal{R}(\bar{x})$. As we discussed in Section (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 either $G^1 = 0$ or if $(\bar{x}, \hat{y}) \in \mathcal{F}$ (typically, we would require this), 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.

### 2.5.2 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).

### 2.5.3 Weighted Sums Heuristic

This heuristic employs 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 degrading the other while maintaining feasibility Ehrgott and Wiecek [2005]) of the following problem by using a weighted-sum subproblem [Geoffrion, 1968].

$$\text{argvmin}_{(x,y)\in\mathcal{S}}\{cx + d^1y, d^2y\}, \tag{12}$$

where the operator argvmin means finding a solution (or more than one) that is efficient. For more details, see [DeNegre, 2011].

# 3 Software Framework

In this section, we describe the implementation of the `MibS` software [DeNegre et al., 2017]. This paper refers to version `0.95.1`, the latest released version at the time of this writing. The overall algorithm employed in `MibS` combines the procedures described in Section 2 in a fashion typical of existing branch-and-cut algorithms for other problems. In fact, `MibS` is built on top of the BLIS solver originally built for parallel MILP [Xu et al., 2009]. There are a number of important parameters, described in Section 3.1 below, that determine how the various components described in Section 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 and understanding their role in the algorithm is crucial to understanding the overall strategy, described in Section 3.2. In Section 3.3, we describe important implementational issues surrounding how the second-level problem is solved. Finally, in Section 3.4, we describe the class structure of the C++ code that encapsulates the implementation.

## 3.1 Parameters

A wide range of parameters are available for controlling the algorithm.

**Branching strategy.** There are several parameters that control branching strategy. The main one is `branchStrategy`, which controls what variables we are allowed to branch on. 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 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 setting uses `pseudocost` scoring to select the final candidate.

**Search strategy.** The search strategy used by `MibS` is controlled by ALPS, the underlying tree search framework. The default is a standard hybrid best first and diving 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 impossible for `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 Section 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 repetitive calculation 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 be able to efficiently check for membership.

**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 Section 3.3).

- `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 $x_L = x_L^t$ 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 $[x_L^t, \texttt{secondLevelIsInfeasible}]$ can added to the linking solution pool $\mathcal{L}$.
- If (SL-MILP) has an optimal solution, we can avoid solving problem (SL-MILP) for any $(x, y)$ for which $x_L = x_L^t$ arising in the future by adding the tuple $[x_L^t, \texttt{secondLevelIsFeasible}]$ 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 $(x^t, y^t) \in X \times Y$, $x_L^t \notin \mathcal{L}$ (we have not previously solved (SL-MILP) for $x_L^t$), 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 $x_L = x_L^t$ (if such solution exists).

- `computeBestUBWhenLVarsFixed`: Whether to solve when $l_{x_L}^t = u_{x_L}^t = x_L^t$.
- `computeBestUBWhenLVarsInt`: Whether to solve when $x_L^t \in \mathbb{Z}^L$.
- `computeBestUBWhenXVarsInt`: Whether to solve when $x^t \in X$.

After solving (UB), we know that no solution $(x, y)$ with $x_L = x_L^t$ 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 solutions $(x, y)$ arising in the future for which $x_L = x_L^t$, the tuple $[x_L^t, \texttt{secondLevelIsFeasible}]$ should be replaced with the tuple $[x_L^t, \texttt{UBIsSolved}]$ 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 $x_L^t$) when `branchStrategy` is `linking`, all linking variables are fixed, and $(x^t, y^t) \in X \times Y$. This does not mean that in this case, solving problem (UB) (and further fathoming node $t$) is the only algorithmic option.

### 3.2 Outline of Algorithm

Algorithm 1 gives the general outline of the node processing loop in this branch-and-cut algorithm. Note that to avoid making the algorithm complicated, the algorithm as stated assumes that `useLinkingSolutionPool` parameter is set to `true`, however, in `MibS`, the option of not using linking solution pool is also provided. At a high-level, the procedure consists of

1. Solve the relaxation (LR$^t$) (line 3) and prune the node (lines 5–6) if
   - (LR$^t$) is infeasible; or
   - $L^t \geq U$; or
   - $x_L$ is fixed and it has been stored in set $\mathcal{L}$ with either `secondLevelIsInfeasible` or `UBIsSolved` tags.
2. If (SL-MILP) was not previously solve 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 $\Rightarrow$ $x^t \notin \mathcal{F}_1$ and we add $[x_L^t, \texttt{secondLevelIsInfeasible}]$ to $\mathcal{L}$ (lines 9–10).
   - (SL-MILP) is feasible$\Rightarrow$ We add $[x_L^t, \texttt{secondLevelIsFeasible}]$ to $\mathcal{L}$ (lines 11–12).
3. If (SL-MILP) was solved now or previously and is feasible(line 13), we have either
   - $(x^t, y^t) \in \mathcal{F} \Rightarrow$ update $U$ and fathom node $t$. (lines 15–17).
   - $(x^t, y^t) \notin \mathcal{F}$
     - update $U$ if $(x^t, \hat{y}^t) \in \mathcal{F}$(in case of not solving (UB)) (lines 25–26) and eliminate $(x^t, y^t)$ (lines 27–32).
     - 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 18–19).
       * (UB) is feasible $\Rightarrow$ update $U$ (lines 20–21).
       * Remove $[x_L^t, \texttt{UBIsSolved}]$ from set $\mathcal{L}$ and add $[x_L^t, \texttt{UBIsSolved}]$ (lines 22).
       * If $x_L$ is fixed, fathom node $t$ (lines 23–24).
4. Finally, we must either branch or remove $(x^t, y^t)$ by adding valid inequalities (lines 27–32).
   - If there are no branching candidates, then we must remove $(x^t, y^t)$ by adding valid inequalities (lines 27–28).
   - If (SL-MILP) was not solved now or previously, we must branch if $(x^t, y^t) \in X \times Y$ (lines 29–30).
   - Otherwise, we have the choice of either adding valid inequalities or branching (lines 31–32).

**Algorithm 1:** Node processing loop in `MibS`

**Input :** [Set $\mathcal{L}, U$]
**Output:** [Set $\mathcal{L}, U, L^t, U^t$]

`branch` $\leftarrow$ `False`, $L^t \leftarrow -\infty$, $U^t \leftarrow \infty$
**while** `branch` is `False` **do**
  Solve (LR$^t$)
  $L^t \leftarrow$ The optimal value of (LR$^t$)
  **if** (LR$^t$) is infeasible **or** $L^t \geq U$ **or**
  ($x_L$ is fixed **and** ($[x_L^t,$ `secondLevelIsInfeasible`$] \in \mathcal{L}$ **or** $[x_L^t,$ `UBIsSolved`$] \in \mathcal{L}$)) **then**
    Fathom node $t$
  **if** $[x_L^t, \cdot] \notin \mathcal{L}$ **and**
  ((`branchStrategy` is `linking` **and** $(x^t, y^t) \in X \times Y$ **and** $x_L$ is fixed) **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** $x_L$ is fixed)) **then**
    Solve (SL-MILP) with $x_L = x_L^t$
    **if** (SL-MILP) is infeasible **then**
      $\mathcal{L} \leftarrow \mathcal{L} \cup [x_L^t,$ `secondLevelIsInfeasible`$]$
    **else**
      $\mathcal{L} \leftarrow \mathcal{L} \cup [x_L^t,$ `secondLevelIsFeasible`$]$
  **if** $[x_L^t,$ `secondLevelIsFeasible`$] \in \mathcal{L}$ **or** $[x_L^t,$ `UBIsSolved`$] \in \mathcal{L}$ **then**
    $\hat{y}^t \leftarrow$ The optimal solution of (SL-MILP)
    **if** $(x^t, y^t) \in \mathcal{F}$ **then**
      $U^t \leftarrow cx^t + d^1 y^t$, $U \leftarrow \min\{U, U^t\}$
      Fathom node $t$
    **if** $[x^t,$ `UBIsSolved`$] \notin \mathcal{L}$ **and**
    ((`branchStrategy` is `linking` **and** $(x^t, y^t) \in X \times Y$ **and** $x_L$ is fixed) **or**
    (`computeBestUBWhenXVarsInt` **and** $x^t \in X$) **or**
    (`computeBestUBWhenLVarsFixed` **and** $x_L$ is fixed)**or**
    (`computeBestUBWhenLVarsInt`)) **then**
      Solve (UB)
      **if** (UB) is feasible **then**
        $U^t \leftarrow$ Optimal value of (UB), $U \leftarrow \min\{U, U^t\}$
      $\mathcal{L} \leftarrow \mathcal{L} \setminus [x_L^t,$ `secondLevelIsFeasible`$]$, $\mathcal{L} \leftarrow \mathcal{L} \cup [x_L^t,$ `UBIsSolved`$]$
      **if** $x_L$ is fixed **then**
        Fathom node $t$
    **else if** $(x^t, \hat{y}^t) \in \mathcal{F}$ **then**
      $U^t \leftarrow cx^t + d^1 \hat{y}^t$, $U \leftarrow \min\{U, U^t\}$
  **if** ((`branchStrategy` is `linking` **and** $x_L$ is fixed) **or** `branchStrategy` is `fractional`)
  **and** $(x^t, y^t) \in X \times Y$ **then**
    Remove $(x^t, y^t)$ by generating a cut
  **else if** $[x_L^t, \cdot] \notin \mathcal{L}$**and** $(x^t, y^t) \in X \times Y$ **then**
    `branch` $\leftarrow$ `True`
  **else**
    Remove $(x^t, y^t)$ by generating a cut or `branch` $\leftarrow$ `True`

### 3.3 Solving the Second-level Problem

It is evident that much (if not most) of the computational effort in executing the algorithm we have just described arises from the time required to solve (SL-MILP) and (UB). A major focus of ongoing work, therefore, is the development of methodology to reduce this running time.

In closely related work on solving two-stage stochastic mixed integer programs, methodology for warm-starting the solution process of an MILP using information derived from previously solved instances has been developed. Hassanzadeh and Ralphs [2014] 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 in 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 $\phi$. This method of solving such a sequence of MILPs has been implemented within the SYMPHONY MILP solver [Ralphs et al., 2016, Ralphs and Güzelsoy, 2006, 2005], 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 [CPLEX] and Cbc [Forrest, 2017a].

### 3.4 Design of `MibS`

In this section, we briefly describe the overall design of the software. `MibS` is an open-source implementation in C++ of the branch-and-cut algorithm described in Section 3.2. 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 [COIN-OR]. These packages include the following.

- The COIN-OR High Performance Parallel Search (CHiPPS) Framework [Ralphs et al., 2004], which includes a hierarchy of three libraries.
  - Abstract Library for Parallel Search (ALPS) [Xu et al., 2016a, 2005]: This project is utilized for managing the global branch and bound.
  - Branch, Constrain, and Price Software (BiCePS) [Xu et al., 2016b, Ralphs et al., 2004]: This project provides base classes for objects used in `MibS`.
  - BiCePS Linear Integer Solver (BLIS) [Xu et al., 2016c, 2009]: This project is the parallel MILP Solver framework from which `MibS` is derived.
- COIN-OR Linear Programming Solver (CLP) [Forrest, 2017b]: `MibS` employs this software for solving the LPs arising in the branch-and-cut algorithm.
- SYMPHONY [Ralphs et al., 2016, Ralphs and Güzelsoy, 2005]: This software is used for solving the MILPs required to be solved in the branch-and-cut algorithm.
- COIN-OR Cut Generation Library (CGL) [Cgl]: Both `MibS` and SYMPHONY utilize this library to generate valid inequalities valid for MILPs.
- COIN-OR Open Solver Interface (OSI) [Osi]: 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 derived from BlisModel class, gets the information of original problem from the input files which consist of an MPS file and an auxiliary information file.
- MibSBilevel: This class is specific to MibS and is utilized for checking the bilevel feasibility of solutions of relaxation problem. Furthermore, this class finds the bilevel feasible solutions as described in lines 16, 21 and 26 of Algorithm 1.
- MibSTreeNode: This is a derived class from base class BlisTreeNode and is used for processing the nodes.
- MibSCutGenerator: This class is specific to MibS and is employed to generate the cuts described in Section 2.4.
- MibSBranchStrategyXyz: These classes (one for each strategy MibS can use 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 which is specific to MibS, is used for generating heuristic solutions by employing the primal heuristics illustrated in Section 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 being easy to use, we have tried to make MibS user friendly as much as possible. Prior to starting the process of solving a problem, MibS investigates different properties of the problem such as type of the problem (interdiction or general), type of the variables (continuous, discrete or binary) and signs of the coefficient matrices. Then, it checks the parameters set by the user and modifies them if it specifies that those values of parameters are not valid for this problem. However, it informs the user in case of changing any of the parameters. For example, MibS turns off the cuts which are selected by the user, but are not valid for the problem.

# 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.

Testing of the effectiveness of various classes of valid inequalities and parameters for controlling them is left for 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 [2011], 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. [2016a], which are zero-sum IBLPs ($d^1 = -d^2$) with binary variables and all constraints at the second level.
- `MIBLP-XU`: This set was introduced in [Xu and Wang, 2014] and includes 100 randomly-generated instances. In these problems, all first-level variables are integer 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 $0.4n_1$.

Table 1 summarizes the properties of the data sets.

Table 1: The summary of data setse

| 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 | 24 |
| MIBLP-XU | 10-460 | 10-460 | 4-184 | 4-184 | discrete | continuous, discrete | 100 |

All computational results we report were run on a Linux (Debian 8.7) operating system running 16 AMD processors on a 800 MHz speed and 32 GB RAM. The CPU-time limit was set to 3600 seconds, 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 MILP cuts by the Cut Generation Library [Cgl] and primal heuristics were disabled. SYMPHONY was employed as the MILP solver while the preprocessing and primal heuristics were turned-off. 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.

All instance were initially solved by all methods described later, but to plot performance profiles, we chose the 121 problems that can be solved by at least one method in 3600 seconds and whose solution time exceeds 5 seconds for at least one method. This test set was used for all plots shown.

### 4.1 Impact of Strategy for Solving (SL-MILP) and (UB)

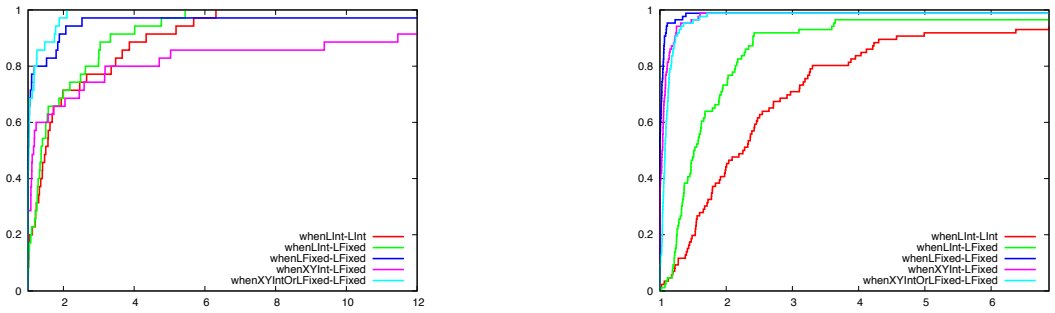
In order to evaluate the impact of the parameters for solving problems (SL-MILP) and (UB), we employed five different methods:

- `whenLInt-LInt`: Problems (SL-MILP) and (UB) are both solved only when $x_L^t \in \mathbb{Z}$.
- `whenLInt-LFixed`: Problem (SL-MILP) is solved when $x_L^t \in \mathbb{Z}$ and problem (UB) is solved when all linking variables are fixed.
- `whenLFixed-LFixed`: Problems (SL-MILP) and (UB) are both solved only when linking variables are fixed.
- `whenXYInt-LFixed`: Problem (SL-MILP) is solved only when $(x^t, y^t) \in X \times Y$ and problem (UB) is solved only when, in addition, all linking variables are fixed.
- `whenXYIntOrLFixed-LFixed`: Problem (SL-MILP) is only solved when $(x^t, y^t) \in X \times Y$ or all linking variables are fixed and problem (UB) is solved only whenever all linking variables are fixed.

In this first set of experiments, the `branchStrategy` parameter was set to `linking` and `useLinkingSolutionPool` was set to `true`.

The performance profiles shown in Figure 2 compare the solution time of the five described methods. Figure 2a shows the results of `IBLP-DEN` and `IBLP-FIS` sets and Figure 2b is corresponding to the results of `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 of these methods. 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.




(a) `IBLP-DEN` and `IBLP-FIS` sets (b) `MIBLP-XU` set

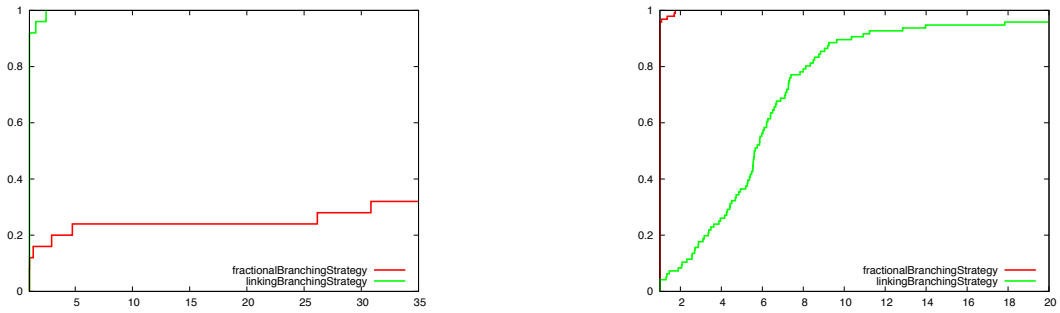
Figure 2: Impact of the parameters for solving problems (SL-MILP) and (UB).

## 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 ef-

fect 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 $r_1 \leq r_2$ (25 instances) and $r_1 > r_2$ (96 instances). Figure 3 shows the performance profiles for these two sets with the solution time as the performance measure.




(a) $r_1 \leq r_2$ (b) $r_1 > r_2$

Figure 3: Impact of the `branchStrategy` parameter.

Figure 3a shows that, on this set of instances, `linking` performed better when $r_1 \leq r_2$, while Figure 3b indicates that, for these instances, 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.

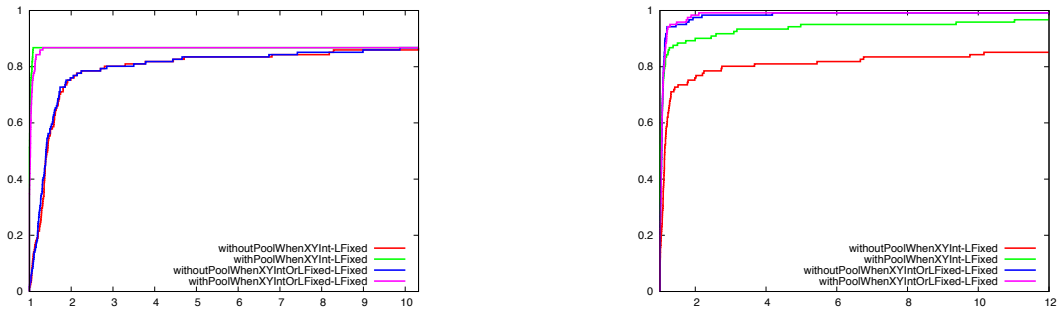
### 4.3 Impact of Linking Solution Pool

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

Each of the above four settings was tested with both `fractional` and `linking` branching strategies, since the linking pool was only expected to have a large impact when branching on non-linking variables is allowed. 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 Figure 4 compare the solution time of the described methods. Figure 4a shows the methods which use `fractional` branching strategy, and Figure 4b shows the methods with `linking` branching strategy.




(a) `fractional` branching strategy

(b) `linking` branching strategy

Figure 4: Impact of the linking solution pool.

As we 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` and `withoutPoolWhenXYInt-LFixed` for both the `fractional` and `linking` branching strategies; and
- `withPoolWhenXYIntOrLFixed-LFixed` and `withoutPoolWhenXYIntOrLFixed-LFixed` for `fractional` branching strategy.

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

# 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

T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

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

E. Balas and R. Jeroslow. Canonical cuts on the unit hypercube. *SIAM Journal on Applied Mathematics*, 23(1):61–69, 1972. ISSN 00361399. URL http://www.jstor.org/stable/2099623.

J. F. Bard. Practical bilevel optimization: algorithms and applications. *Springer Science & Business Media*, 30, 2013.

J. F. Bard and J. T. Moore. An algorithm for the discrete bilevel programming problem. *Naval Research Logistics*, 39(3):419–435, 1992.

J. Bracken and J. T. McGill. Mathematical programs with optimization problems in the constraints. *Operations Research*, 21(1):37–44, 1973.

A. Caprara, M. Carvalho, A. Lodi, and G. J. Woeginger. Bilevel knapsack with interdiction constraints. *INFORMS Journal on Computing*, 28(2):319–333, 2016.

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

Cgl. Cut generator library, 2017. URL https://projects.coin-or.org/Cgl.

COIN-OR. Computational Infrastructure for Operations Research, 2017. URL https://www.coin-or.org.

M. Conforti, G. Cornuéjols, and G. Zambelli. *Integer programming*, volume 271. Springer, 2014.

CPLEX. Ibm ilog cplex optimizer, 2017. URL https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.

S. T. DeNegre. *Interdiction and Discrete Bilevel Linear Programming*. Phd, Lehigh University, 2011. URL http://coral.ie.lehigh.edu/{~}ted/files/papers/ScottDeNegreDissertation11.pdf.

S. T. DeNegre and T. K. Ralphs. A branch-and-cut algorithm for bilevel integer programming. In *Proceedings of the Eleventh INFORMS Computing Society Meeting*, pages 65–78, 2009. doi: 10.1007/978-0-387-88843-9_4. URL http://coral.ie.lehigh.edu/~ted/files/papers/BILEVEL08.pdf.

S. T. DeNegre, T. K. Ralphs, and S. Tahernejad. Mibs version 0.95, 2017.

M. Ehrgott and M. M. Wiecek. Multiobjective programming. In M. Ehrgott, J. Figueira, and S. Greco, editors, *Multiple Criteria Decision Analysis: State of the Art Surveys*, pages 667–722. Springer, Berlin, Germany, 2005.

N. P. Faísca, V. Dua, B. Rustem, P. M. Saraiva, and E. N. Pistikopoulos. Parametric global optimisation for bilevel programming. *Journal of Global Optimization*, 38:609–623, 2007.

M. Fischetti, I. Ljubić, M. Monaci, and M. Sinnl. Intersection cuts for bilevel optimization. Technical report, 2016a.

M. Fischetti, I. Ljubic, M. Monaci, and M. Sinnl. A new general-purpose algorithm for mixed-integer bilevel linear programs. Technical report, 2016b.

J. J. Forrest. Coin-or branch and cut, 2017a. URL https://projects.coin-or.org/Cbc.

J. J. Forrest. Coin-or lp solver, 2017b. URL https://projects.coin-or.org/Clp.

A. M. Geoffrion. Proper efficiency and the theory of vector maximization. *Journal of Mathematical Analysis and Applications*, 22:618–630, 1968.

A. Hassanzadeh and T. K. Ralphs. A generalized benders' algorithm for two-stage stochastic program with mixed integer recourse. Technical report, COR@L Laboratory Report 14T-005, Lehigh University, 2014. URL http://coral.ie.lehigh.edu/~ted/files/papers/SMILPGenBenders14.pdf.

M. Hemmati and C. Smith. A mixed integer bilevel programming approach for a competitive set covering problem. Technical report, Clemson University, 2016.

P.-M. Kleniati and C. S. Adjiman. Branch-and-sandwich: A deterministic global optimization algorithm for optimistic bilevel programming problems. part i: Theoretical development. *Journal of Global Optimization*, 60:425–458, 2014a.

P.-M. Kleniati and C. S. Adjiman. Branch-and-sandwich: A deterministic global optimization algorithm for optimistic bilevel programming problems. part ii: Convergence analysis and numerical results. *Journal of Global Optimization*, 60:459–481, 2014b.

H. Marchand, A. Martin, R. Weismantel, and L. Wolsey. Cutting planes in integer and mixed integer programming. *Discrete Applied Mathematics*, 123(1):397–446, 2002.

A. Mitsos. Global solution of nonlinear mixed integer bilevel programs. *Journal of Global Optimization*, 47:557–582, 2010.

J. T. Moore and J. F. Bard. The mixed integer linear bilevel programming problem. *Operations research*, 38(5):911–921, 1990.

Osi. Open solver interface, 2017. URL https://projects.coin-or.org/Osi.

T. K. Ralphs and M. Güzelsoy. The symphony callable library for mixed-integer linear programming. In *Proceedings of the Ninth INFORMS Computing Society Conference*, pages 61–76, 2005. doi: 10.1007/0-387-23529-9_5. URL http://coral.ie.lehigh.edu/~ted/files/papers/SYMPHONY04.pdf.

T. K. Ralphs and M. Güzelsoy. Duality and warm starting in integer programming. In *The Proceedings of the 2006 NSF Design, Service, and Manufacturing Grantees and Research Conference*, 2006. URL http://coral.ie.lehigh.edu/~ted/files/papers/DMII06.pdf.

T. K. Ralphs, L. Ladányi, and M. J. Saltzman. A library hierarchy for implementing scalable parallel search algorithms. *Journal of Supercomputing*, 28:215–234, 2004. doi: 10.1023/B:SUPE.0000020179.55383.ad. URL http://coral.ie.lehigh.edu/~ted/files/papers/JSC02.pdf.

T. K. Ralphs, M. Güzelsoy, and A. Mahajan. Symphony version 5.6, 2016.

L. Vicente, G. Savard, and J. Júdice. Discrete linear bilevel programming problem. *Journal of Optimization Theory and Applications*, 89(3):597–614, 1996.

H. Von Stackelberg. *Marktform und Gleichgewicht*. Julius Springer, 1934.

U.-P. Wen and A. Huang. A simple tabu search method to solve the mixed-integer linear bilevel programming problem. *European Journal of Operational Research*, 88:563–571, 1996.

R. Wollmer. Removing arcs from a network. *Operations Research*, 12(6):934–940, 1964.

K. Wolter. Implementation of Cutting Plane Separators for Mixed Integer Programs. Master's thesis, Technische Universität Berlin, 2006.

P. Xu. *Three Essays on Bilevel Optimization and Applications*. PhD thesis, Iowa State University, 2012.

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

Y. Xu, T. K. Ralphs, L. Ladányi, and M. J. Saltzman. Alps: A framework for implementing parallel search algorithms. In *The Proceedings of the Ninth INFORMS Computing Society Conference*, pages 319–334, 2005. doi: 10.1007/0-387-23529-9_21. URL http://coral.ie.lehigh.edu/~ted/files/papers/ALPS04.pdf.

Y. Xu, T. K. Ralphs, L. Ladányi, and M. J. Saltzman. Computational experience with a software framework for parallel integer programming. *The INFORMS Journal on Computing*, 21:383–397, 2009. doi: 10.1287/ijoc.1090.0347. URL http://coral.ie.lehigh.edu/~ted/files/papers/CHiPPS-Rev.pdf.

Y. Xu, T. K. Ralphs, L. Ladányi, and M. J. Saltzman. Alps version 1.5, 2016a.

Y. Xu, T. K. Ralphs, L. Ladányi, and M. J. Saltzman. Biceps version 0.94, 2016b.

Y. Xu, T. K. Ralphs, L. Ladányi, and M. J. Saltzman. Blis version 0.94, 2016c.

B. Zeng and Y. An. Solving bilevel mixed integer program by reformulations and decomposition. *Optimization On-line*, 2014.