A diving heuristic for mixed-integer problems with unbounded semi-continuous variables

Katrin Halbig [*a] Alexander Hoen [b] Ambros Gleixner [bc]
Jakob Witzig [d] Dieter Weninger [e]

October 2024

Abstract

Semi-continuous decision variables arise naturally in many real-world applications. They are defined to take either value zero or any value within a specified range, and occur mainly to prevent small nonzero values in the solution. One particular challenge that can come with semi-continuous variables in practical models is that their upper bound may be large or even infinite. In this article, we briefly discuss these challenges, and present a new diving heuristic tailored for mixed-integer optimization problems with general semi-continuous variables. The heuristic is designed to work independently of whether the semi-continuous variables are bounded from above, and thus circumvents the specific difficulties that come with unbounded semi-continuous variables. We conduct extensive computational experiments on three different test sets, integrating the heuristic in an open-source MIP solver. The results indicate that this heuristic is a successful tool for finding high-quality solutions in negligible time. At the root node the primal gap is reduced by an average of 5 % up to 21 %, and considering the overall performance improvement, the primal integral is reduced by 2 % to 17 % on average.

Keywords: Integer programming · Semi-continuous variables · Indicator constraints · Diving heuristic · Supply chain management

1 Introduction

In mathematical optimization, semi-continuous variables arise naturally in models for many real-world applications, and have been studied since at least 1979 [32]. Formally, a variable y is called *semi-continuous* if it is defined to take either the

*Corresponding author
[a]Friedrich-Alexander-Universität Erlangen-Nürnberg, Department of Data Science, Cauerstr. 11, 91058 Erlangen, Germany, katrin.halbig@fau.de
[b]Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany, hoen@zib.de
[c]Hochschule für Technik und Wirtschaft Berlin, 10313 Berlin, Germany, gleixner@htw-berlin.de
[d]SAP SE, Dietmar-Hopp-Allee 17, 69190 Walldorf, Germany, jakob.witzig@sap.com
[e]Friedrich-Alexander-Universität Erlangen-Nürnberg, Department of Mathematics, Cauerstr. 11, 91058 Erlangen, Germany, dieter.weninger@fau.de

value 0 or any value within the range specified by its *lower semi-continuous bound* s and *upper semi-continuous bound* u:

$$y \in \{0\} \cup [s, u] \text{ with } 0 < s \leq u \text{ and } u \in \mathbb{R}_+ \cup \{\infty\}. \tag{1}$$

This characteristic makes semi-continuous variables a powerful modeling tool for capturing scenarios where a variable's influence is either fully absent or continuous within a specific range.

Semi-continuous variables were introduced to simplify the solution of blending problems when materials must be excluded from the blend if they cannot be used in significant quantities [8]. Nowadays, they are used in a wide range of real-world applications. In portfolio optimization, semi-continuous variables are often used to avoid a large number of small trades or holdings which are undesirable because of management and transaction costs [36]. Also in unit commitment problems in electrical power systems, semi-continuous variables are used to model minimum online and offline times of units [16]. Supply chain management problems are also problems in which semi-continuous variables often occur. Here, semi-continuous variables are used to model economic order quantities, technical requirements because transportation of low quantities is undesirable from an operational point of view, or to describe the state of a machine that is either turned off and thus produces nothing or turned on and thus has to produce at least a minimal amount (so-called *minimum lot size*) due to economic and technical reasons; see [24, 28, 35, 43].

In the present article, we distinguish between two cases of semi-continuous variables. If u is finite, we say the variable is *bounded semi-continuous* and if u is infinite, we refer to it as an *unbounded semi-continuous* variable. In particular, we are interested in the common scenario where semi-continuous variables are part of a mixed-integer linear optimization problem. Formally, we consider an optimization problem of the form

$$\begin{aligned}
\min_{x,y} \quad & c^\top x + \sum_{j=1}^{\nu} f_j(y_j) && && \text{(2a)}\\
\text{s.t.} \quad & A(x, y) \geq b, && && \text{(2b)}\\
& y_j \in \{0\} \cup [s_j, u_j], && \text{for all } j \in \mathcal{J}, && \text{(2c)}\\
& x_i \in \mathbb{Z}, && \text{for all } i \in \mathcal{I}_x, && \text{(2d)}\\
& y_i \in \mathbb{Z}, && \text{for all } i \in \mathcal{I}_y, && \text{(2e)}
\end{aligned}$$

where y denotes a vector of semi-continuous variables with lower semi-continuous bound $s \in \mathbb{R}^\nu_+$ and upper bound $u \in (\mathbb{R}_+ \cup \{\infty\})^\nu$. Further variables are summarized in vector x. The objective function is defined by vector $c \in \mathbb{R}^\eta$, which relates to x, and by discontinuous functions

$$f_j \colon \{0\} \cup [s_j, u_j] \to \mathbb{R}, \;\; f_j(y_j) \coloneqq \begin{cases} 0, & \text{if } y_j = 0,\\ d_j y_j + e_j, & \text{if } y_j \geq s_j, \end{cases}$$

which relate to y_j for all $j \in \mathcal{J} \coloneqq \{1, \dots, \nu\}$. Here, $d_j \in \mathbb{R}$ models unit costs and $e_j \in \mathbb{R}$ models set-up costs. Constraint (2c) specifies all ν semi-continuous variables. Further linear constraints are given by a constraint matrix $A \in \mathbb{R}^{m \times (\eta + \nu)}$ with right-hand side vector $b \in \mathbb{R}^m$. Sets $\mathcal{I}_x \subseteq \{1, \dots, \eta\}$ and

$\mathcal{I}_y \subseteq \mathcal{J}$ specify the integer variables. Note that variables y_j with $j \in \mathcal{I}_y$ are also called *semi-integer*, but for the sake of simplicity, we will not distinguish these cases.

In order to motivate that a heuristic approach is appropriate to solve problem (2), let us briefly discuss its complexity. The optimization problem (2) can be transformed into the related decision problem

$$\exists\ (x, y) \text{ satisfying (2b)-(2e) with value } c^\top x + \sum_{j=1}^{\nu} f_j(y_j) \leq k,\ k \in \mathbb{Z}. \tag{3}$$

This decision problem (3) is $\mathcal{NP}$-hard, because the decision problem 0-1 integer program (see [47]) is polynomially reducible to (3) by not using any variables x and requiring $s_j = u_j = 1$ and $e_j = 0$ for all $j \in \mathcal{J}$. Since the decision problem (3) is no more difficult to solve than the optimization problem (2), it is evident that problem (2) is intractable even for finite u.

Solution approaches for mixed-integer problems with semi-continuous variables include, for example, strengthening of semi-continuous bounds [4, 48], branch-and-bound methods taking into account properties of semi-continuous variables [8, 21], generation of semi-continuous cuts [21], convex hull descriptions of problems with semi-continuous variables [5, 22], and heuristics [19, 29]. It is notable that, with the exception of [5, 21, 22], the vast majority of publications on semi-continuous variables is limited to the bounded case. An extensive literature survey of solution methods with a focus on nonlinear optimization problems with semi-continuous variables is provided in [42].

Our contributions include, in Section 2, a discussion of the particular challenges of solving MIPs with unbounded semi-continuous variables, which arise when using an LP relaxation. In Section 3 we propose a diving heuristic that is independent of the boundedness of the semi-continuous variables. The peculiarity of this heuristic is that it performs a depth-first-search in the branch-and-bound tree that indirectly takes into account the characteristics of semi-continuous variables by treating indicator reformulations of the semi-continuous variables appropriately, which means that the way the heuristic works can also be seen as a transfer of the branching approach from [8, 21]. Moreover, a high-performance publicly available C implementation of this heuristic is provided in form of a tight integration with the open-source solver SCIP [12]. Computational results thereof are discussed in Section 4. One of our test sets is comprised of real-world supply chain management instances, the other two test sets are publicly available and contain general MIPs and artificially generated supply chain management instances. A brief summary and a short discussion of open research questions for future work in Section 5 conclude the article.

2 Formulations and LP Relaxation

In this section, we will discuss formulations of semi-continuous variables in problem (2), paying particular attention to the differences between bounded and unbounded semi-continuous variables.

Consider the feasible set $\{0\} \cup [s, u] \subseteq \mathbb{R}$ of one semi-continuous variable. This set is the union of two polyhedra. For formulations of semi-continuous variables, it is now crucial how the convex hull of the union of the two polyhedra can be represented. For $u = \infty$ the convex hull is not a *bounded MIP-representable* set [30, 44], which means that no linear mixed-binary formulation exists. If $u < \infty$, then the situation is less difficult since both polyhedra have the same recession cones and thus the convex hull of their union can itself be formulated as a polyhedron [7, 18].

If unbounded semi-continuous variables occur in problem (2), we are confronted with a more complex problem structure than if only bounded semi-continuous variables occur. Therefore, the question arises as to whether it is necessary to consider a problem with unbounded semi-continuous variables at all. In Section 4 we substantiate that instances from practice actually contain unbounded semi-continuous variables. In many cases, this is simply due to the fact that no feasible finite upper bound is known, and determining a bound is of no interest to the practitioner since it would (a) not be a practical concern if y takes a large value, and (b) it is known that unboundedness of y does not render the entire problem unbounded.

Even more, it is sometimes not possible to determine a bound small enough to be used in numerical computations without knowing a near-optimal solution. For example, in strategic supply chain planning, which involves optimizing operations over a period ranging from six months to multiple years, lot sizes are typically unbounded. Given the extended time horizon, time steps are discretized into daily, weekly, or monthly intervals. The feasibility of setting a maximum lot size (that is, upper bound u) largely depends on the nature of the goods being produced, transported, or procured, as well as their respective units of measurement. In some cases, expanding capacity to accommodate larger lot sizes can be straightforward, such as by leasing additional trucks. However, imposing finite maximum lot sizes can lead to numerical instability due to the substantial quantities involved within each time interval. Consequently, modelers often omit maximum lot sizes from their models when deemed unnecessary or impractical. We therefore need to develop methods that can deal with unbounded semi-continuous variables *during* the solving process.

There are several approaches to formulating semi-continuous variables. Customary are the *complementary formulation* resulting in non-convex non-linear problems [9, 41] and *disjunctive programming* approaches [6, 15]. The *big-M-formulation* [15, 45] is presumably the easiest and commonly used disjunctive programming approach, and is applicable in the case of a finite upper bound u. By using a constant M with $M \geq u$ a bounded semi-continuous variable can be formulated with an additional binary variable z as

$$y \leq M \cdot z, \tag{4a}$$
$$s \cdot z \leq y \leq u, \tag{4b}$$
$$z \in \{0, 1\}. \tag{4c}$$

For $z = 1$ the constraint (4a) is redundant and $s \leq y \leq u$ is active. In the other case $z = 0$, the variable y is fixed to 0 by (4a) and (4b) whereby $y \leq u$ is redundant. In the case of an infinite upper bound u one can use a sufficiently large positive constant M, such that no optimal solution is cut off.

However, finding a suitable M can be a major difficulty, see [31] for a related discussion. In practice, a pragmatic strategy is sometimes adopted by setting M to a "huge" value, even though this may cut off optimal solutions.

Furthermore, the big-M method may cause numerical problems. When using a large value for M, the reciprocal $1/M$ can be close to the machine accuracy or the tolerances used by mathematical programming solvers when working with floating point arithmetic and become indistinguishable from zero. This carries the risk of numerical difficulties during the solution process. For example, if z takes a value close to zero, $M \cdot z$ can still be large enough to set y to a substantial value, whereas z is evaluated as integer feasible and zero. In addition, the solutions of continuous relaxations may be weak, i.e., very far away from the optimal solution value. Some of these difficulties can be defused by strengthening the M value within the branch-and-bound tree by separation of local cuts [17] and bound propagation techniques [4, 39].

Because of the above disadvantages the focus of this article is on *indicator constraints* [9, 15], which allows us to model bounded as well as unbounded semi-continuous variables exactly. Indicator constraints activate a linear inequality based on the state of a binary variable known as the *indicator variable*. When the indicator variable is set to a certain value, the corresponding constraint is included in the model; otherwise, it is ignored. This representation is used henceforth and also in our later test sets, and has the advantage of being supported by virtually all state-of-the-art solvers.

To express a semi-continuous variable y using indicator constraints, an additional binary indicator variable z is introduced. This indicator variable controls the constraint $y \leq 0$ (5a). If $z = 0$, it activates the constraint and forces the value of y to be 0. To ensure that $y \geq s$ holds when $z = 1$, an additional constraint $y \geq s \cdot z$ is introduced (5b). Overall, we obtain the formulation

$$z = 0 \implies y \leq 0, \tag{5a}$$
$$s \cdot z \leq y \leq u, \tag{5b}$$
$$z \in \{0, 1\}. \tag{5c}$$

By reformulating all semi-continuous variables y_j for all $j \in \mathcal{J}$ in problem (2) with approach (5), we arrive at

$$\begin{aligned}
\min_{x,y,z} \quad & c^\top x + d^\top y + e^\top z && && \text{(6a)}\\
\text{s.t.} \quad & A(x, y) \geq b, && && \text{(6b)}\\
& z_j = 0 \implies y_j \leq 0, && \text{for all } j \in \mathcal{J}, && \text{(6c)}\\
& s_j \cdot z_j \leq y_j \leq u_j, && \text{for all } j \in \mathcal{J}, && \text{(6d)}\\
& z_j \in \{0, 1\}, && \text{for all } j \in \mathcal{J}, && \text{(6e)}\\
& x_i \in \mathbb{Z}, && \text{for all } i \in \mathcal{I}_x, && \text{(6f)}\\
& y_i \in \mathbb{Z}, && \text{for all } i \in \mathcal{I}_y, && \text{(6g)}
\end{aligned}$$

where c, d, e, A, b, $\mathcal{J}$, $\mathcal{I}_x$, and $\mathcal{I}_y$ are defined as in Section 1.

Branch-and-bound type algorithms [20, 33] rely on solving a relaxation to obtain information for bounding and branching decisions. An LP relaxation is derived from a MIP by ignoring the integrality conditions on the integer variables. If the LP relaxation is used to solve MIPs with a branch-and-bound enumeration approach, this is referred to as the *LP-based branch-and-bound* method [2, 34]. For an LP relaxation to a MIP involving indicator constraints as problem (6), the integrality conditions (6e), (6f), (6g), and the indicator constraint (6c) are ignored. Therefore, the semi-continuous variable y can take any value in $[0,u]$, which often results in weak LP relaxations [9]. In particular, if objective coefficient e is positive, z can be set to zero to reduce set-up costs, but y can be arbitrarily large in $[0,u]$. We note that this LP relaxation is even worse than the LP relaxation of the big-M-formulation. Due to numerical issues and the possible cutoff of optimal solutions, we nevertheless choose the formulation with indicator constraints in this article.

3 A Tailored Diving Heuristic

The determination of feasible solutions is essential for branch-and-bound type algorithms [2, 20, 33] in order to cut off parts of the enumeration tree and thus accelerate the solution process. In Section 1 we have shown that problem (3) is $\mathcal{NP}$-Hard and thus it is not surprising that heuristics play a central role in solving problem (6).

One class of heuristics are the so-called *diving heuristics* (or *dive-and-fix heuristics*). Starting with a solution of a relaxation at the current node in a branch-and-bound tree diving heuristics strengthen variable bounds and reoptimize the relaxation iteratively. This simulates a depth-first-search to a leaf in the branch-and-bound tree. Diving heuristics for mixed-integer linear programs (MIP) are described, for example, in [10, 46, 47], and for mixed-integer nonlinear programs (MINLP) in [14].

We now propose the diving heuristic *Indicator Diving* (ID), which aims to determine a feasible solution for problem (6). Indicator Diving can get called at any node in a branch-and-bound tree after solving the LP relaxation of problem (6) with solution $(\widehat{x},\widehat{y},\widehat{z})$. If the LP solution is feasible for (6), there is nothing to do, and the heuristic is not called. Otherwise, one has to specify a set $\mathcal{K}$ of candidate variables that are responsible for the infeasibility of $(\widehat{x},\widehat{y},\widehat{z})$. These are at first all integer variables x_i, $i \in \mathcal{I}_x$, y_i, $i \in \mathcal{I}_y$, and z_j, $j \in \mathcal{J}$, with fractional value. In our specific case of MIPs with indicator constraints, $\mathcal{K}$ additionally contains all binary indicator variables z_j, if their corresponding indicator constraint is violated as well as z_j is not fixed already and $\widehat{z}_j$ is integer. As long as the indicator variables are unfixed, the indicator constraints are omitted in the LP relaxation and thus the LP solution may violate these constraints.

For every candidate $\kappa \in \mathcal{K} \subseteq \mathcal{I}_x \cup \mathcal{I}_y \cup \mathcal{J}$ three different functions get called:

- a *score* function $\psi\colon \mathcal{K} \to \mathbb{R}$,
- a *rounding* function $\sigma\colon \mathcal{K} \to \{\texttt{up}, \texttt{down}\}$, and
- a *bound value* function $\beta\colon \mathcal{K} \to \mathbb{R}$.

These functions are described in detail later in this section.

In Algorithm 1 the basic operation of Indicator Diving for MIPs with semi-continuous variables as formulated in (6) is presented. The diving heuristic repeatedly selects a candidate $\kappa \in \mathcal{K}$ with maximal score $\psi(\kappa)$, and sets new bounds for the corresponding variable according to the results of $\sigma(\kappa)$ and $\beta(\kappa)$. Afterwards, the LP gets optimized again with the updated bounds. This step is optional and can be skipped at some or all iterations to keep the algorithm efficient. Before updating the LP, it is also possible to propagate bound changes, that is, one tries to utilize the bound change of the candidate variable to strengthen further variable bounds. After updating the LP, one can, in addition, try to find an integral solution via rounding [3].

Algorithm 1: Diving Heuristic for MIPs with semi-cont. variables

```
Input: Problem (6) with LP relaxation, LP feasible solution $(\widehat{x}, \widehat{y}, \widehat{z})$,
       score function $\psi$, rounding function $\sigma$, bound value function $\beta$.
Output: Feasible solution of (6) if one has been found.
$\mathcal{K} := \{j \in \mathcal{J} \mid \widehat{z}_j \in \mathbb{Z},\ z_j$ unfixed, (6c) violated$\}$
    $\cup \{i \in \mathcal{I}_x \mid \widehat{x}_i \notin \mathbb{Z}\} \cup \{i \in \mathcal{I}_y \mid \widehat{y}_i \notin \mathbb{Z}\} \cup \{j \in \mathcal{J} \mid \widehat{z}_j \notin \mathbb{Z}\}$
while $\mathcal{K} \neq \emptyset$ do
    for $\kappa \in \mathcal{K}$ do
        Calculate score $\psi(\kappa)$, rounding direction $\sigma(\kappa)$ and new value
          $\beta(\kappa)$ of bound.
    end
    Select candidate $\kappa$ with maximal score.
    $\mathcal{K} \leftarrow \mathcal{K} \setminus \{\kappa\}$
    if $\sigma(\kappa) =$ up then
        Increase lower bound to $\beta(\kappa)$.
    else
        Decrease upper bound to $\beta(\kappa)$.
    end
    (Optional) propagate bound change.
    (Optional) update and solve LP.
    if LP infeasible then
        Backtrack or abort.
    else
        Update $\mathcal{K}$ based on new LP solution.
    end
    (Optional) round LP solution.
end
```

If the LP is infeasible, one can backtrack by undoing the last bound change and selecting the opposite rounding direction, or abort the heuristic without finding a feasible solution. If the LP solution is feasible for the original problem (6), the candidate set $\mathcal{K}$ is empty and the heuristic terminates with this feasible solution.

In order to describe the heuristic Indicator Diving completely, we define now the three mentioned functions ψ, σ, and β. For the score function ψ we have to distinguish between indicator variables and integer variables. If we consider an indicator variable z_κ, the score is given by

$$\psi(\kappa) := \begin{cases} -1, & \text{if } \widehat{y}_\kappa \in \{0\} \cup [s_\kappa, u_\kappa], \\ 100 \cdot (s_\kappa - \widehat{y}_\kappa)/s_\kappa, & \text{if } \widehat{y}_\kappa \in (0, s_\kappa). \end{cases} \tag{7}$$

For all other candidate variables we use another already existing diving strategy. For example, one can switch to Farkas Diving [46], as we will do for the computational tests in Section 4. Since indicator variables should always be preferred, the score for other candidate variables is scaled to a range less than -1. For example, one can apply a sigmoid function, which has an "S"-shaped graph, to map the scores to the interval $[-300, -100]$ while preserving the order, see [27, 40]. As the candidate variable with the highest score is taken next, the indicator variables with semi-continuous variables outside their domain are treated first, followed by the indicator variables with semi-continuous variables inside their domain and thus actually already fulfilled, and finally, all other integer candidate variables are treated.

Moreover, the functions σ and β are defined as follows: If candidate κ corresponds to an indicator variable z_κ, the rounding function is given by

$$\sigma(\kappa) := \begin{cases} \texttt{up}, & \text{if } \widehat{y}_\kappa \geq 0.5\ s_\kappa, \\ \texttt{down}, & \text{if } \widehat{y}_\kappa < 0.5\ s_\kappa, \end{cases} \tag{8}$$

and the bound value function is given by

$$\beta(\kappa) := \begin{cases} 1, & \text{if } \widehat{y}_\kappa \geq 0.5\ s_\kappa, \\ 0, & \text{if } \widehat{y}_\kappa < 0.5\ s_\kappa. \end{cases} \tag{9}$$

In other words, the binary indicator variable is fixed to one—and thus semi-continuous variable y_κ is fixed to $[s_\kappa, u_\kappa]$—if the value of the corresponding semi-continuous variable in the LP solution is at least 50 % of the lower semi-continuous bound. Otherwise, it is fixed to zero—and thus y_κ is also fixed to zero. At this point, Indicator Diving differs from well-known diving heuristics in the literature, since the bounding step for one variable depends on the LP solution value of another variable. For all other candidate variables, we switch to the same already existing diving heuristic as used for the score function. Typically, such a diving heuristic uses for a candidate variable x_κ (or y_κ, z_κ) bound value $\beta(\kappa) := \lceil \widehat{x}_\kappa \rceil$ if $\sigma(\kappa) = \texttt{up}$ and $\beta(\kappa) := \lfloor \widehat{x}_\kappa \rfloor$ if $\sigma(\kappa) = \texttt{down}$.

4 Computational Study

In this section, we present a comprehensive computational study of the diving heuristic proposed in Section 3. After describing the experimental setup, results for two main experiments are shown, for the first only the root node is processed and the second analyzes the overall performance impact.

Computational setup. For the experiments, we used a pre-release version of SCIP 9.0 [13] (git hash 7cc9c068bc) with SoPlex [13] version 6.0.3 (git hash 555f5d54) as underlying LP solver, running single-threaded. The code is publicly available at `https://github.com/scipopt`. Indicator Diving is added as SCIP heuristic plugin `heur_indicatordiving.c` using the generic diving algorithm framework of SCIP. The output stream has been modified slightly in some cases to retrieve the desired information.

Roughly summarized, SCIP begins the solving process of a problem with a presolving phase to simplify the problem, for example, fixing variables or deleting redundant constraints. The root node is then created and selected as the first open node. After bound propagation is performed, the LP relaxation of the node is solved, potentially through multiple rounds in which cutting planes are added. If the node is not solved to integer optimality, branching is performed to create new child nodes. SCIP continues by selecting an open node and repeating this process until the problem is solved or predefined solving limits are reached.

During this process, primal heuristics are typically invoked at two key points for each node: before solving the LP relaxation and after solving the LP relaxation. Since Indicator Diving requires an LP solution, it is called after solving the LP relaxation, and for our experiments, it is invoked only at the root node.

All presented computational results were generated on a compute cluster using compute nodes with Intel Xeon Gold 6326 processors with 2.9 GHz and 32 GB RAM; see [23] for more details.

Performance metrics. For the purpose of evaluating the performance of Indicator Diving, we compare *shifted geometric means* of *primal gaps* and *primal integrals.*

For a single instance, let $\mathcal{R}_{max}$ be the total running time and $\mathcal{R}_1, \dots, \mathcal{R}_\tau \in [0, \mathcal{R}_{max}]$ be the points in time when a new incumbent solution is found, $\mathcal{R}_0 \coloneqq 0$, $\mathcal{R}_{\tau+1} \coloneqq \mathcal{R}_{max}$. Let $(\tilde{x}, \tilde{y}, \tilde{z})_{opt}$ be an optimal or best known solution and let $(\tilde{x}, \tilde{y}, \tilde{z})_{\mathcal{R}}$ be the incumbent solution at point $\mathcal{R} \in [0, \mathcal{R}_{max}]$.

The primal gap function $\mathrm{PG}\colon [0, \mathcal{R}_{max}] \to [0, 1]$ is defined as

$$\mathrm{PG}(\mathcal{R}) \coloneqq \begin{cases} 1, & \text{if no incumbent until point } \mathcal{R}, \\ 0, & \text{if } |(c,d,e)^\top(\tilde{x}, \tilde{y}, \tilde{z})_{opt}| = |(c,d,e)^\top(\tilde{x}, \tilde{y}, \tilde{z})_{\mathcal{R}}| = 0, \\ 1, & \text{if } (c,d,e)^\top(\tilde{x}, \tilde{y}, \tilde{z})_{opt} \cdot (c,d,e)^\top(\tilde{x}, \tilde{y}, \tilde{z})_{\mathcal{R}} < 0, \\ \dfrac{|(c,d,e)^\top(\tilde{x}, \tilde{y}, \tilde{z})_{opt} - (c,d,e)^\top(\tilde{x}, \tilde{y}, \tilde{z})_{\mathcal{R}}|}{\max\{|(c,d,e)^\top(\tilde{x}, \tilde{y}, \tilde{z})_{opt}|, |(c,d,e)^\top(\tilde{x}, \tilde{y}, \tilde{z})_{\mathcal{R}}|\}}, & \text{else,} \end{cases} \tag{10}$$

and building on this the primal integral is defined as

$$\mathrm{PI} \coloneqq \sum_{t=1}^{\tau+1} \mathrm{PG}(\mathcal{R}_{t-1}) \cdot (\mathcal{R}_t - \mathcal{R}_{t-1}). \tag{11}$$

The primal integral provides an absolute measurement for the performance of primal heuristics by considering the evolution of the incumbent solution over time, rewarding algorithms that find good solutions early. For a detailed description of the primal integral see [11]. For the best-known solution of an instance as the base value we can use publicly available solutions (see [49]) and/or results of previous runs.

For average values over multiple instances, we use the *shifted geometric mean* (SGM) of these values over all instances. For values $w_1, \dots, w_N \geq 0$ and shift $s \geq 0$ we determine the SGM by

$$\mathrm{sgm}(w_1, \dots, w_N, s) \coloneqq \left(\prod_{i=1}^{N} (w_i + s) \right)^{\frac{1}{N}} - s. \tag{12}$$

Unlike the arithmetic mean, the shifted geometric mean has the advantage that extreme values have less influence on the average. For further discussion on evaluating computational results with SGMs see, for example, [1]. If not stated otherwise, we use for all average values the shifted geometric mean with shift $s = 1$.

Test sets. We consider three sets of instances. The first test set contains 588 real-world supply chain management instances (SCM) with semi-continuous variables modeled with indicator constraints supplied by our industry partner SAP SE [37]. The most important components are inventory holding, capacity restrictions, procurement, transport, production, and demand fulfillment. In [25] a more detailed description of very similar instances can be found. The second test set contains 192 artificially generated supply chain management instances based on real-world supply chain management models, again with semi-continuous variables modeled by indicator constraints. They represent a fictive company procuring components, producing cellphones of different types, transporting them to distribution centers, and satisfying customer demands. This test set is provided by SAP SE [37] and is publicly available at [38]. We refer to it hereinafter as Cellphone. The third set contains all 42 instances of the MIPLIB 2017 [26] Collection with indicator constraints, publicly available at [49]. We have not filtered these further, so they do not necessarily conform to our specifications, but necessary ones are checked when Indicator Diving is called.

Performance variability can be caused by the compute nodes as well as various random factors within SCIP. For example, the score function (7) for indicator variables with a semi-continuous variable within its domain (which is constant -1) is also equipped with a random factor in our implementation. This mechanism obtains the highest score (among a series of constants -1) regardless of the input sequence of the problem to SCIP. To account for the effect of performance variability, we use three different seeds (including the default seed of zero) and treat every instance-seed combination as one individual observation. From now on, we refer to such a combination simply as an instance, which triples the number of instances. So, we have test set SCM with 1764 instances, Cellphone with 576 instances, and Miplib with 126 instances.

In Table 1 some basic statistics of all three test sets are summarized. We consider only instances that reached the calling point of ID, that is, instances for which the root LP was solved and ID was indeed called. Grouped by test set, different numbers of the instances are reported, once from the original problem and once from the problem that ID has received. In row “vars” and “conss” we count the number of all variables and all constraints, in row “ind” we count only the number of indicator constraints. The latter number is further divided into the number of indicator constraints belonging to unbounded semi-continuous variables (“unbd”) and to bounded semi-continuous variables with very large upper bound ($10^4 < u < \infty$, “big”). The line was drawn at 10^4, since SCIP adds an additional linear inequality $y \leq uz$ for smaller bounds; see also (4a) for the big-M formulation. In column “total” we state the sum of these counts over all instances per test set. The minimum, maximum, and shifted geometric mean of these figures is reported in columns “min”, “max”, and “sgm”, respectively.

As an example, for test set SCM 1300 instances reached the calling point. These instances have between 7 and 29599 indicator constraints in the original problem with an average of 78.94 indicator constraints. In the received problem the average number of indicator constraints reduces to 67.74. It is noteworthy that in test set Cellphone in the original problem all indicator constraints belong to unbounded semi-continuous variables, but in the received problem all indicator constraints belong to bounded semi-continuous variables, due to presolving processes which determine finite bounds. Although there are no or only

		original problem				received problem			
		total	min	max	sgm	total	min	max	sgm
SCM (#1300)	vars	14420569	1034	594067	6090.87	8345699	345	229155	4119.30
	conss	8481884	287	515321	2329.60	4007191	166	225346	1493.93
	ind	362939	7	29599	78.94	209101	2	24083	67.74
	unbd	315083	0	29599	76.75	276	0	46	0.02
	big	47856	0	23754	0.05	104519	0	23844	5.45
Cellphone (#463)	vars	6778402	7065	24123	13498.18	75880	32	493	124.13
	conss	4119674	4407	14467	8241.05	67253	27	439	109.75
	ind	144149	187	415	298.43	15614	4	107	24.55
	unbd	144149	187	415	298.43	0	0	0	0.00
	big	0	0	0	0.00	14162	4	96	22.25
Miplib (#99)	vars	3417906	1923	315484	20765.78	2081992	626	304610	8797.25
	conss	4564677	3912	268835	33283.11	2435727	2153	165383	16217.48
	ind	525174	52	21247	2856.09	414986	24	46395	1883.76
	unbd	0	0	0	0.00	0	0	0	0.00
	big	0	0	0	0.00	0	0	0	0.00

Table 1: Statistics on dimensions of the instances.

a few unbounded indicator constraints remaining in the test sets, indicator constraints are nonetheless a concern because many have still a big upper bound, leading to the issues discussed in Section 2. Note also that in test set Miplib all indicator constraints are already bounded by a small bound in the original problem. Therefore, strictly speaking, they do not exhibit the property of containing big or infinite upper bounds, which motivated the design of Indicator Diving.

Performance results. We start the evaluation with a root node experiment, for which we stop SCIP after the end of the root node or a time limit of 5 h is reached. Within this solution process, Indicator Diving was called, if possible, after solving the LP relaxation, along with other default heuristics of SCIP. Calling ID is, for example, not possible if all indicator constraints were fixed or deleted due to presolving processes.

	#instances			PG root			PG heur
	called	found	best found	with ID	w/o ID	ratio	
SCM (#1764)	1300	1052	287	0.15	0.19	0.79	0.30
Cellphone (#576)	463	365	55	0.23	0.25	0.92	0.46
Miplib (#126)	99	16	7	0.76	0.80	0.95	0.50

Table 2: Root node experiment.

For test set SCM, Indicator Diving was called on 1300 out of 1764 instances and found a feasible solution in 1052 cases. Since diving heuristics in SCIP use an objective cutoff, all these solutions improve the primal bound at the point in time they are found. In 287 cases the solution found by ID was still the best solution at the end of the root node.

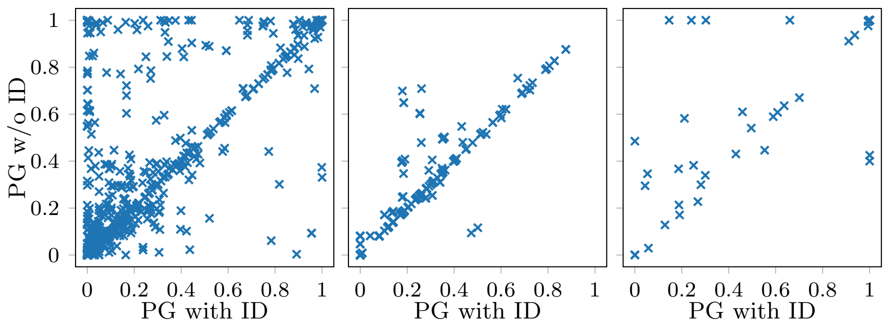
These figures are summarized for all three test sets in Table 2. Moreover, we take a look at the primal gap after solving the root node (column "PG root") for all instances on which ID was called. We calculate the shifted geometric mean of the relative primal gaps and compare it to a run without ID. For SCM the primal gap is reduced by 21% on average, from 0.19 to 0.15. The average reduction of the primal gap at the point in time when the solution of ID is passed to SCIP is given in the last column "PG heur". The latter figures take only instances on which ID found a solution into account.

Taking a look at Table 2, we can conclude that on SCM and Cellphone Indicator Diving has a high probability to find an improving solution. In addition, over all test sets ID yields a large reduction of up to 70 % of the primal gap at the point in time when the solution is passed to SCIP. This reduction of the primal gap persists until the end of the root node, and is large, especially on SCM, reaching up to 21 % on average.

In addition, we aim to verify that indicator constraints modeling semi-continuous variables are, in fact, problematic and that they are frequently violated by the LP relaxation solution. To this end, for the two test sets SCM and Cellphone, we investigated how many indicator variables were actually diving candidates in the first iteration of Algorithm 1. For test set SCM, 13 % of the indicator variables were diving candidates whereby 60 % of the associated semi-continuous variables were in the prohibited range $(0, s)$. Considering only semi-continuous variables that have a big or infinite upper bound, 8 % of the related indicator variables were diving candidates and of these 50 % of the semi-continuous variables were in the range $(0, s)$. For Cellphone, 14 % of the indicator variables were diving candidates whereby 99 % of the associated semi-continuous variables were in $(0, s)$. Restricted to semi-continuous variables that have a big or infinite upper bound, the numbers are essentially the same.

Furthermore, in Figure 1 we compare the final primal gap achieved with and without ID over all instances for which ID was called. The horizontal axis indicates the primal gap at the end of the root node computation for the run with ID, and the vertical axis indicates the same value in the run without ID. Thus, crosses in the left upper corner represent instances with a reduction of the primal gap. As crosses may overlap, we explicitly report the number of instances with a strict improvement (Wins) and a strict degradation (Losses) in the captions. We consider an improvement/degradation to be strict if the absolute difference is at least 10^{-4}. These numbers demonstrate that there are considerably more instances with a strict improvement of the primal gap. It is also noteworthy that within SCM there are many instances where the gap decreases remarkably from a large value to almost zero due to Indicator Diving.

Finally, we analyze the overall performance impact of Indicator Diving on the complete tree search. For this we use a time limit of 20 minutes and no node limit. The run with Indicator Diving is compared to a run without Indicator Diving. The results are summarized in Table 3 for each test set separately. We disregard the instances where ID was not called at all, and consider the subsets of instances on which ID was called, and the subsets on which it was called and found a solution. The numbers of instances of these subsets are stated in column "#inst" and are slightly different compared to the numbers in the root node experiment above. This is because we have different time limits and because



(a) SCM (#1300)
Wins: 432
Losses: 154

(b) Cellphone (#463)
Wins: 106
Losses: 20

(c) Miplib (#99)
Wins: 17
Losses: 8

Figure 1: Primal gap at end of root node computation.

ID can get called more than once due to restarts in SCIP. We state the ratio of the shifted geometric means of the primal integrals of both runs in column “ratio PI” whereby a value less than 1 indicates a performance improvement, and the average time spent in ID (in seconds) is stated in column “time ID”. Moreover, the average solving time of the run with ID and the average solving time of the run without ID is stated as well as the ratio thereof. In the last two columns the number of instances that reached the time limit is specified.

					solving time			timeout	
		#inst	ratio PI	time ID	with ID	w/o ID	ratio	with ID	w/o ID
SCM (#1764)	called	1304	0.86	0.080	29.1	28.9	1.01	277	262
	found	1064	0.83	0.056	18.9	18.9	1.00	137	129
Cellphone (#576)	called	463	0.97	0.004	161.1	177.0	0.91	281	289
	found	425	0.98	0.004	143.2	158.7	0.90	248	257
Miplib (#126)	called	94	0.98	4.456	375.3	380.2	0.99	51	51
	found	19	0.92	1.087	77.5	79.2	0.98	3	4

Table 3: Performance comparison of SCIP with and without ID. Absolute times are given in seconds.

Taking a look at Table 3, we can conclude that Indicator Diving yields a large reduction of the primal integral between 14 % and 17 % on test set SCM and a reduction between 2 % and 8 % on the other two test sets. Moreover, ID uses only a negligible proportion of the total solving time on all three test sets. The impact on the overall solving time is almost neutral on SCM and Miplib, but reduced by up to 10 % on Cellphone. This aligns with previous findings that the impact of primal heuristics on the total solving time is minor [11]. Timeouts indicate that there is a considerable number of hard instances in our test sets even though the average solving time is moderate. Their variation depends more on performance variability of the subsequent processes of SCIP than on ID.

To summarize, Indicator Diving helps the MIP solver to find better solutions earlier during the solving process, which is an important, if not the most important metric in practice. As the total solving time is neutral or increases, we observe that its impact on performance is somewhat leveled out by the ensemble of other solving techniques applied by SCIP.

5 Conclusion

In this article, we discuss the challenges of using bounded and unbounded semi-continuous variables and propose a tailored diving heuristic for solving mixed-integer problems with semi-continuous variables that can be employed either standalone or integrated into a MIP solver. An implementation in C of this heuristic and two of the three treated test sets are publicly accessible to be able to follow the extensive computational experiments in detail. One of the test sets was newly generated by our industry partner in the course of this research work and made available to the public in order to facilitate future research on the problems studied.

As part of this work, two computational experiments were carried out to evaluate the practical suitability of the diving heuristic. The first one is a root node experiment, which exhibits a reduction of the primal gap on all three test sets when using the heuristic. Particularly noteworthy is the result on one test set, where a reduction of the primal gap of 21 % is achieved. In a second experiment we compared the performance of a MIP solver with and without the new heuristic and observed that the use of the heuristic improves the primal integral by 2 % to 17 %.

Finally, we would like to address two research questions that complement the topic of the article presented. First, we have found that unbounded semi-continuous variables occur in real-world instances and we have also shown in our computational study that unbounded semi-continuous variables are often transformed to bounded semi-continuous variables by bound propagation methods. This leads to the assumption that specially adapted and more aggressively used domain reduction methods and presolving techniques might be able to transform even more unbounded semi-continuous variables into bounded semi-continuous variables. Such approaches might also make it possible to determine tighter bounds for bounded semi-continuous variables. Second, if an LP relaxation, in particular a simplex tableau, is used when generating cuts for problem (6), then the indicator constraints are not taken into account. Consequently, important information is lost in the cut generating process, which usually leads to weaker cuts. It would now be an interesting research question what possibilities there are to incorporate the information of the indicator constraints into the generation of valid cuts.

Acknowledgments

The work for this article has been partly conducted within the Research Campus Modal funded by the German Federal Ministry of Education and Research (BMBF grant number 05M20ZBM). The authors would like to thank SAP for

its long-term support and for providing test instances.

References

[1] T. Achterberg. *Constraint Integer Programming.* PhD thesis, Technische Universität Berlin, 2007.

[2] T. Achterberg. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009. doi:https://doi.org/10.1007/s12532-008-0001-1.

[3] T. Achterberg, T. Berthold, and G. Hendel. Rounding and propagation heuristics for mixed integer programming. In D. Klatte, H.-J. Lüthi, and K. Schmedders, editors, *Operations Research Proceedings 2011*, pages 71–76, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-29210-1. doi:https://doi.org/10.1007/978-3-642-29210-1_12.

[4] T. Achterberg, R. E. Bixby, Z. Gu, E. Rothberg, and D. Weninger. Presolve reductions in mixed integer programming. *INFORMS Journal on Computing*, 32(2):473–506, 2020. doi:https://doi.org/10.1287/ijoc.2018.0857.

[5] G. Angulo, S. Ahmed, and S. S. Dey. Semi-continuous network flow problems. *Mathematical Programming*, 145(1):565–599, Jun 2014. ISSN 1436-4646. doi:https://doi.org/10.1007/s10107-013-0675-7.

[6] E. Balas. Disjunctive programming. In P. Hammer, E. Johnson, and B. Korte, editors, *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 3–51. Elsevier, 1979. doi:https://doi.org/10.1016/S0167-5060(08)70342-X. URL `https://www.sciencedirect.com/science/article/pii/S016750600870342X`.

[7] E. Balas. On the convex hull of the union of certain polyhedra. *Operations Research Letters*, 7(6):279–283, 1988. ISSN 0167-6377. doi:https://doi.org/10.1016/0167-6377(88)90058-2. URL `https://www.sciencedirect.com/science/article/pii/0167637788900582`.

[8] E. M. L. Beale. Integer programming. In K. Schittkowski, editor, *Computational Mathematical Programming*, pages 1–24, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg. ISBN 978-3-642-82450-0.

[9] P. Belotti, P. Bonami, M. Fischetti, A. Lodi, M. Monaci, A. Nogales-Gómez, and D. Salvagnin. On handling indicator constraints in mixed integer programming. *Computational Optimization and Applications*, 65(3): 545–566, Dec 2016. ISSN 1573-2894. doi:https://doi.org/10.1007/s10589-016-9847-8.

[10] T. Berthold. Heuristics of the branch-cut-and-price-framework SCIP. In J. Kalcsics and S. Nickel, editors, *Operations Research Proceedings 2007*, pages 31–36, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-77903-2. doi:https://doi.org/10.1007/978-3-540-77903-2_5.

[11] T. Berthold. Measuring the impact of primal heuristics. *Operations Research Letters*, 41(6):611–614, 2013. ISSN 0167-6377. doi:https://doi.org/10.1016/j.orl.2013.08.007. URL `https://www.sciencedirect.com/science/article/pii/S0167637713001181`.

[12] K. Bestuzheva, M. Besançon, W.-K. Chen, A. Chmiela, T. Donkiewicz, J. van Doornmalen, L. Eifler, O. Gaul, G. Gamrath, A. Gleixner, L. Gottwald, C. Graczyk, K. Halbig, A. Hoen, C. Hojny, R. van der Hulst, T. Koch, M. Lübbecke, S. J. Maher, F. Matter, E. Mühmer, B. Müller, M. E. Pfetsch, D. Rehfeldt, S. Schlein, F. Schlösser, F. Serrano, Y. Shinano, B. Sofranac, M. Turner, S. Vigerske, F. Wegscheider, P. Wellner, D. Weninger, and J. Witzig. Enabling research through the scip optimization suite 8.0. *ACM Trans. Math. Softw.*, 49(2), jun 2023. ISSN 0098-3500. doi:https://doi.org/10.1145/3585516.

[13] S. Bolusani, M. Besançon, K. Bestuzheva, A. Chmiela, J. Dionísio, T. Donkiewicz, J. van Doornmalen, L. Eifler, M. Ghannam, A. Gleixner, C. Graczyk, K. Halbig, I. Hedtke, A. Hoen, C. Hojny, R. van der Hulst, D. Kamp, T. Koch, K. Kofler, J. Lentz, J. Manns, G. Mexi, E. Mühmer, M. E. Pfetsch, F. Schlösser, F. Serrano, Y. Shinano, M. Turner, S. Vigerske, D. Weninger, and L. Xu. The SCIP Optimization Suite 9.0. Technical report, Optimization Online, February 2024. URL `https://optimization-online.org/?p=25734`.

[14] P. Bonami and J. P. M. Gonçalves. Heuristics for convex mixed integer nonlinear programs. *Computational Optimization and Applications*, 51(2):729–747, 2012. doi:https://doi.org/10.1007/s10589-010-9350-6.

[15] P. Bonami, A. Lodi, A. Tramontani, and S. Wiese. On mathematical programming with indicator constraints. *Mathematical Programming*, 151(1):191–223, Jun 2015. ISSN 1436-4646. doi:https://doi.org/10.1007/s10107-015-0891-4.

[16] H. Chen, M. Liu, Y. Cheng, and S. Lin. Modeling of unit commitment with ac power flow constraints through semi-continuous variables. *IEEE Access*, 7:52015–52023, 2019. doi:10.1109/ACCESS.2019.2911632.

[17] V. Chvátal, W. J. Cook, and D. G. Espinoza. Local cuts for mixed-integer programming. *Math. Program. Comput.*, 5(2):171–200, 2013. doi:https://doi.org/10.1007/s12532-013-0052-9.

[18] M. Conforti, G. Cornuéjols, and G. Zambelli. Polyhedral approaches to mixed integer linear programming. In M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, editors, *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pages 343–385. Springer, 2010. doi:https://doi.org/10.1007/978-3-540-68279-0_11.

[19] I. Crévits, S. Hanafi, R. Mansi, and C. Wilbaut. Iterative semi-continuous relaxation heuristics for the multiple-choice multidimensional knapsack problem. *Computers & Operations Research*, 39(1):32–41, Jan 2012. ISSN 0305-0548. URL `https://www.sciencedirect.com/science/article/pii/S0305054811000037`.

[20] R. J. Dakin. A tree-search algorithm for mixed integer programming problems. *The Computer Journal*, 8(3):250–255, 01 1965. ISSN 0010-4620. doi:https://doi.org/10.1093/comjnl/8.3.250.

[21] I. R. de Farias. Semi-continuous cuts for mixed-integer programming. In D. Bienstock and G. Nemhauser, editors, *Integer Programming and Combinatorial Optimization*, pages 163–177, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-25960-2.

[22] I. R. de Farias and M. Zhao. A polyhedral study of the semi-continuous knapsack problem. *Mathematical Programming*, 142(1):169–203, Dec 2013. ISSN 1436-4646. doi:https://doi.org/10.1007/s10107-012-0566-3.

[23] Erlangen National High Performance Computing Center (NHR@FAU) . Woody throughput cluster (Tier3), last accessed on 2024-01-29. URL `https://hpc.fau.de/systems-services/documentation-instructions/clusters/woody-cluster/`.

[24] D. Erlenkotter. Ford whitman harris and the economic order quantity model. *Operations Research*, 38(6):937–946, 1990. ISSN 0030364X, 15265463. URL `http://www.jstor.org/stable/170961`.

[25] G. Gamrath, A. Gleixner, T. Koch, M. Miltenberger, D. Kniasew, D. Schlögel, A. Martin, and D. Weninger. Tackling industrial-scale supply chain problems by mixed-integer programming. *Journal of Computational Mathematics*, 37:866 – 888, 2019. doi:https://doi.org/10.4208/jcm.1905-m2019-0055.

[26] A. Gleixner, G. Hendel, G. Gamrath, T. Achterberg, M. Bastubbe, T. Berthold, P. M. Christophel, K. Jarck, T. Koch, J. Linderoth, M. Lübbecke, H. D. Mittelmann, D. Ozyurt, T. K. Ralphs, D. Salvagnin, and Y. Shinano. MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. *Mathematical Programming Computation*, 2021. doi:https://doi.org/10.1007/s12532-020-00194-3.

[27] K. Halbig, A. Göß, and D. Weninger. Exploiting user-supplied decompositions inside heuristics. Technical report, Optimization Online, May 2024. URL `https://optimization-online.org/?p=23386`.

[28] F. W. Harris. How many parts to make at once. *Factory, The Magazine of Management*, 10(2):135–136, 1913.

[29] F. Hooshmand. A hybrid dc algorithm for an optimization problem with semi-continuous variables and cardinality constraint. In P. Vasant, G.-W. Weber, J. A. Marmolejo-Saucedo, E. Munapo, and J. J. Thomas, editors, *Intelligent Computing & Optimization*, pages 914–925, Cham, 2023. Springer International Publishing. ISBN 978-3-031-19958-5.

[30] R. G. Jeroslow and J. K. Lowe. Modelling with integer variables. In B. Korte and K. Ritter, editors, *Mathematical Programming at Oberwolfach II*, pages 167–184, Berlin, Heidelberg, 1984. Springer Berlin Heidelberg. ISBN 978-3-642-00915-0. doi:https://doi.org/10.1007/BFb0121015.

[31] T. Kleinert, M. Labbé, F. Plein, and M. Schmidt. Technical note—there's no free lunch: On the hardness of choosing a correct big-M in bilevel optimization. *Operations Research*, 68(6):1716–1721, 2020. URL `https://EconPapers.repec.org/RePEc:inm:oropre:v:68:y:2020:i:6:p:1716-1721`.

[32] A. Land and S. Powell. Computer codes for problems of integer programming. In P. Hammer, E. Johnson, and B. Korte, editors, *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 221–269. Elsevier, 1979. doi:https://doi.org/10.1016/S0167-5060(08)70352-2. URL `https://www.sciencedirect.com/science/article/pii/S0167506008703522`.

[33] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960. ISSN 00129682, 14680262. URL `http://www.jstor.org/stable/1910129`.

[34] J. T. Linderoth and M. W. P. Savelsbergh. A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing*, 11(2):173–187, 1999. doi:https://doi.org/10.1287/ijoc.11.2.173.

[35] Y. W. Park and D. Klabjan. Lot sizing with minimum order quantity. *Discrete Applied Mathematics*, 181:235–254, 2015. ISSN 0166-218X. doi:https://doi.org/10.1016/j.dam.2014.09.015. URL `https://www.sciencedirect.com/science/article/pii/S0166218X1400417X`.

[36] A. F. Perold. Large-scale portfolio optimization. *Management Science*, 30(10):1143–1160, 1984. ISSN 00251909, 15265501. URL `http://www.jstor.org/stable/2631383`.

[37] SAP SE. SAP Software Solutions — Business Applications and Technology, last accessed on 2024-02-29. URL `https://www.sap.com`.

[38] SAP SE or an SAP affiliate company. MILP Benchmarks CellphoneCo., Jan. 2023. URL `https://github.com/SAP-samples/ibp-sop-benchmarks-milp-cellphoneco`. last accessed on 2024-02-29.

[39] M. W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *INFORMS J. Comput.*, 6(4):445–454, 1994. doi:https://doi.org/10.1287/ijoc.6.4.445.

[40] L. Schewe, M. Schmidt, and D. Weninger. A Decomposition Heuristic for Mixed-Integer Supply Chain Problems. *Operations Research Letters*, 48(3):225–232, 2020. ISSN 0167-6377. doi:https://doi.org/10.1016/j.orl.2020.02.006.

[41] O. Stein, J. Oldenburg, and W. Marquardt. Continuous reformulations of discrete–continuous optimization problems. *Computers & Chemical Engineering*, 28(10):1951–1966, 2004. ISSN 0098-1354. doi:https://doi.org/10.1016/j.compchemeng.2004.03.011. URL `https://www.sciencedirect.com/science/article/pii/S0098135404000870`. Special Issue for Professor Arthur W. Westerberg.

[42] X. Sun, X. Zheng, and D. Li. Recent advances in mathematical programming with semi-continuous variables and cardinality constraint. *Journal of the Operations Research Society of China*, 1(1):55–77, Mar 2013. ISSN 2194-6698. doi:https://doi.org/10.1007/s40305-013-0004-0.

[43] C. H. Timpe and J. Kallrath. Optimal planning in large multi-site production networks. *European Journal of Operational Research*, 126 (2):422–435, 2000. ISSN 0377-2217. doi:https://doi.org/10.1016/S0377-2217(99)00301-X. URL `https://www.sciencedirect.com/science/article/pii/S037722179900301X`.

[44] J. P. Vielma. Mixed integer linear programming formulation techniques. *SIAM Rev.*, 57(1):3–57, jan 2015. ISSN 0036-1445. doi:https://doi.org/10.1137/130915303.

[45] H. P. Williams. *Model building in mathematical programming.* John Wiley and Sons, 2013. ISBN 9781118443330.

[46] J. Witzig and A. Gleixner. Conflict-driven heuristics for mixed integer programming. *INFORMS Journal on Computing*, 33(2):706–720, 2021. doi:https://doi.org/10.1287/ijoc.2020.0973.

[47] L. A. Wolsey. *Integer Programming.* John Wiley and Sons, Ltd, 2020. ISBN 9781119606475. doi:https://doi.org/10.1002/9781119606475. URL `https://onlinelibrary.wiley.com/doi/book/10.1002/9781119606475`.

[48] H. Zhang and J. Yuan. A combined variable aggregation presolving technique for mixed integer programming. *Operations Research Letters*, 53:107074, 2024. ISSN 0167-6377. doi:https://doi.org/10.1016/j.orl.2024.107074. URL `https://www.sciencedirect.com/science/article/pii/S0167637724000105`.

[49] Zuse Institute Berlin. MIPLIB 2017 – The Mixed Integer Programming Library, last accessed on 2024-02-29. URL `https://miplib.zib.de/`.