A Penalty Branch-and-Bound Method for Mixed-Binary Linear Complementarity Problems

Marianna De Santis, Sven de Vries, Martin Schmidt, Lukas Winkel

Abstract. Linear complementarity problems (LCPs) are an important modeling tool for many practically relevant situations but also have many important applications in mathematics itself. Although the continuous version of the problem is extremely well studied, much less is known about mixed-integer LCPs (MILCPs) in which some variables have to be integer-valued in a solution. In particular, almost no tailored algorithms are known besides reformulations of the problem that allow to apply general-purpose mixed-integer linear programming solvers. In this paper, we present, theoretically analyze, enhance, and test a novel branch-and-bound method for MILCPs. The main property of this method is that we do not “branch” on constraints as usual but by adding suitably chosen penalty terms to the objective function. By doing so, we can either provably compute an MILCP solution if it exists or compute an approximate solution that minimizes an infeasibility measure combining integrality and complementarity conditions. We enhance the method by MILCP-tailored valid inequalities, node selection strategies, branching rules, and warmstarting techniques. The resulting algorithm is shown to clearly outperform two benchmark approaches from the literature.

1. Introduction

The linear complementarity problem (LCP) is the task to find a vector \( z \in \mathbb{R}^n \) that satisfies

\[
\begin{align*}
q + Mz &\geq 0, \\
z^\top(q + Mz) &= 0
\end{align*}
\]

or to show that no such vector exists. The problem is denoted by LCP\((q, M)\). LCPs are an important tool for the modeling and analysis of equilibrium problems in economics, mechanics, and other applied fields, but also have many important applications in mathematics itself. We refer to the seminal textbook by Cottle et al. (2009) for a general overview and to the book by Gabriel, Conejo, Fuller, et al. (2012) for a large collection of applications in energy markets. As defined above, the classic LCP is stated in terms of a continuous variable vector \( z \). In practice, however, one also faces situations in which a subset of variables is restricted to take integer values, i.e., \( z_i \in \mathbb{Z} \) for a given index set \( I \subseteq \{1, \ldots, n\} \). This is relevant for situations in which, e.g., an equilibrium problem (modeled as an LCP) needs to satisfy additional integrality constraints. Other situations include those in which further combinatorial constraints are imposed on the LCP’s solution such as equity-enforcement conditions for network flows or integer production levels. For applications related to mixed-integer LCPs (MILCPs) we refer to, e.g., Gabriel

Date: July 19, 2021.

2020 Mathematics Subject Classification. 90-08, 90Bxx, 90C11, 90C33.

Key words and phrases. Mixed-Integer Programming, Linear Complementarity Problems, Mixed-Integer Linear Complementarity Problems, Branch-and-Bound, Penalty Methods.
Although integer solutions of LCPs have been mentioned first in 1940 by Val (1940), the study of the connection between LCPs and integer programming (IP) started in the late 1980s, when it was shown that LCPs and IP are equivalent (Pardalos 1988; Pardalos and Rosen 1988); see also Pardalos (1994) or Pardalos (1996) for survey articles that also discuss the relation of LCPs and IP problems. Some applications of LCPs with mixed-integer variables are also discussed by Pardalos and Nagurney (1990), where the authors mention, e.g., polymatrix games in pure strategies, general economic equilibria with integer activity levels, or spatial price equilibrium problems with discrete commodities. To the best of our knowledge, Pardalos and Nagurney (1990) also present the first algorithm to solve integer LCPs. However, the idea is simple: Enumerate all continuous LCP solutions and check if any of them are purely integer-valued.

For continuous LCPs two of the classic questions are those of existence and uniqueness of solutions, which are closely related to the study of matrix classes; see, e.g., Cottle et al. (2009) for an overview. For purely integer-valued LCP solutions, the respective matrix class $I$ has been introduced by Chandrasekaran et al. (1998) and Cunningham and Geelen (1998) and necessary as well as sufficient conditions are derived for that a given matrix is in $I$. Chandrasekaran et al. (1998) also discuss the “peeling algorithm” for solving integer LCPs. Further and more recent studies of the existence of integer solutions are given by Dubey and Neogy (2018) and Sumita et al. (2018).

Compared to the larger number of theoretical papers mentioned so far, the literature on algorithms for mixed-integer LCPs is rather sparse. In the context of energy applications, Gabriel, Conejo, Ruiz, et al. (2013) reformulate the mixed-integer LCP as a mixed-integer linear problem (MILP) using disjunctive constraints and big-$M$ constants. Unfortunately, this cannot be seen as a general-purpose method since such big-$M$s are not always available. Gabriel (2017) exploits the relation of complementarity problems to the median function (Gabriel 1998; Gabriel and Moré 1997), which again is used to state a proper MILP to solve the mixed-integer LCP. This approach, however, again relies on the choice of big-$M$ constants. Furthermore, Fomeni et al. (2019a,b) exploit reformulation-linearization techniques (RLT) to solve the mixed-integer LCP. Finally, Gabriel, Leal, et al. (2021) use purely continuous reformulations of MILCPs to solve these problems to local optimality by exploiting general-purpose nonlinear programming solvers. To sum up, almost all known solution approaches are based on a reformulation of the mixed-integer LCP so that state-of-the-art MILP solvers can be used.

Our contribution is the following. We present a novel branch-and-bound method that explicitly exploits the structure of mixed-integer LCPs and that has the following main properties. First, in classic branch-and-bound methods, relaxations are solved and the feasible set is tightened again by branching on, e.g., integer variables or constraints. Consequently, the objective function stays the same in the entire branch-and-bound tree but the constraint set is extended if one goes down the tree starting from the root node. Our branch-and-bound approach for mixed-integer LCPs follows a different rationale. Here, the constraint set stays the same over the entire tree and “branching” is realized by adding suitably chosen penalty terms to the objective, i.e., our objective function grows if we go down the tree starting from the root node. In order to obtain efficiently solvable problems in the nodes of our branch-and-bound tree, we restrict ourselves here to positive semi-definite LCP matrices $M$. Second, for practically relevant instances, the existence of mixed-integer feasible solutions of LCPs is often hard to achieve. Thus, it is crucial for practice to algorithmically
deal with the case of non-existence of solutions as well. Since non-existence is often the case, one is also interested in approximate feasible solutions, i.e., points that minimize a certain infeasibility measure that combines both the violation of integrality conditions as well as of complementarity constraints; see, e.g., Gabriel, Conejo, Ruiz, et al. (2013), where similar relaxations are considered. Our penalty-based branch-and-bound approach is explicitly tailored to deal with the case of non-existence: If a mixed-integer feasible LCP solution exists, our algorithm provably finds such a point—or it computes an approximate feasible solution that minimizes a certain infeasibility measure. Third, our branch-and-bound framework allows to incorporate further algorithmic enhancements. Here, we present mixed-integer LCP specific valid inequalities, discuss tailored node selection strategies, branching rules, and warmstarting techniques. Besides these general-purpose enhancements, it is a dedicated feature of our approach that it can be extended by problem-specific techniques if they are at hand for a concrete application problem. Fourth and finally, we test the penalty branch-and-bound method numerically and compare it with two benchmarks: an MILP reformulation of the mixed-integer LCP using big-Ms as well as a straightforward MIQP reformulation—both solved with the state-of-the-art mixed-integer programming solver Gurobi. It turns out that our method (extended by the above mentioned enhancements) clearly outperforms both benchmark approaches.

The remainder of the paper is organized as follows. In Section 2, we present the problem statement and discuss a reformulation of the mixed-integer LCP that is later needed for the derivation of the penalty branch-and-bound method. The method is described in Section 3, where we also prove the correctness of the method by proving the correctness of the branching as well as of the bounding step. Afterward, in Section 4, we discuss further enhancements of the algorithm to improve its performance, which we evaluate in a numerical study in Section 5. Finally, we conclude the paper in Section 6.

2. Problem Statement and Reformulations

One important tool in the analysis and the resolution of the LCP (1) is the following reformulation as a quadratic problem (QP):

\[
\begin{align*}
\min_{z \in \mathbb{R}^n} & \quad z^T(q + Mz) \\
\text{s.t.} & \quad z \geq 0, \quad q + Mz \geq 0.
\end{align*}
\]

It is easy to see that the LCP (1) has a solution if and only if the QP (2) has an optimal solution with objective function value of zero. The mixed-integer linear complementarity problem (MILCP) that we consider in this paper is the task to find a vector \( z \in \mathbb{R}^n \) that satisfies

\[
\begin{align*}
z & \geq 0, \\
q + Mz & \geq 0, \\
z^T(q + Mz) & = 0, \\
z_i & \in \{0, 1\} \quad \text{for } i \in I,
\end{align*}
\]

or to show that no such vector exists. Here, \( I \subseteq \{1, \ldots, n\} \) is the set of indices for which we require the variable to be binary. We denote this problem by LCP\((q, M, I)\). For the ease of presentation, we restrict ourselves in this paper to mixed-binary LCPs, which is, of course, equivalent to considering mixed-integer LCPs for bounded integers.

Besides these solution approaches from the literature (that we discussed in the introduction and that are all based on reformulating the problem as an MILP), another
straightforward possibility to compute a solution is by solving the reformulation as the mixed-integer quadratic problem (MIQP)

$$\begin{align*}
\min_{z \in \mathbb{R}^n} & \quad z^\top (q + Mz) \\ 
\text{s.t.} & \quad z \geq 0, \quad q + Mz \geq 0, \\
& \quad z_i \in \{0, 1\} \text{ for } i \in I.
\end{align*}$$

Note that this MIQP is bounded from below by 0. Hence, the MILCP (3) has a solution if and only if there is an optimal solution of the MIQP (4) with objective function value of 0. In practice, however, different mathematical challenges arise. The combination of integrality and complementarity conditions may make it unlikely that there exists a solution of the MILCP. Since, thus, the existence of solutions cannot be expected in general, approximate solutions are of interest in many cases. Here, approximate solutions are points that violate the “challenging conditions”, i.e., the integrality conditions as well as the complementarity constraints, as little as possible. This means, we call points \((\rho, \sigma)\)-approximate solutions if they are feasible for the relaxed version

$$\begin{align*}
\min_{z \in \mathbb{R}^n} & \quad z^\top (q + Mz) \\ 
\text{s.t.} & \quad z \geq 0, \quad q + Mz \geq 0, \\
& \quad z_i \leq 1 \quad \text{for } i \in I,
\end{align*}$$

of (3). Here, \(\rho \geq 0\) measures the violation of the complementarity constraints, while \(\sigma_i \geq 0, i \in I\), measure the violation of integrality. Of course, one wants to choose these parameters as small as possible. Obviously, we recover the original MILCP (3) for \(\rho = 0\) and \(\sigma_i = 0, i \in I\). For what follows, we do not consider a relaxation of the linear constraints (3a) and (3b) as they are usually less restrictive compared to the combination of integrality and complementarity conditions of the problem.

In the next section, we propose a nonconvex reformulation of Problem (3) that includes a relaxation of the integrality and complementarity conditions as shown in (5). We use this reformulation to derive an algorithm to solve this reformulation to global optimality. This especially means that if the MILCP is solvable, we compute a solution for the MILCP in finite time. On the other hand, if the MILCP is not solvable, we compute a solution that minimizes a certain measure of violation of the integrality and complementarity conditions. We call the proposed algorithm a “penalty branch-and-bound method” since we do not branch via introducing further inequality constraints to the problem in the parent node but via introducing suitably chosen penalty terms. Thus, while branching, our feasible set stays the same and the objective function is extended by adding the penalty terms, which is exactly the other way around compared to classic branch-and-bound in mixed-integer optimization. All details are presented in the next section. To obtain tractable problems to be solved at every node of the branch-and-bound tree, we restrict ourselves to matrices \(M\) that are positive semi-definite.

### 3. A Penalty Branch-and-Bound Method

We look for approximate solutions of (3), i.e., feasible solutions of (5) for which \(\rho\) and \(\sigma\) are minimized. To this end, we consider an optimization problem in which a combination of the violation of the complementarity conditions (3c) and the violation of the binary conditions (3d) is minimized over the feasible set

$$Z := \{z \in \mathbb{R}^n : z \geq 0, \quad q + Mz \geq 0, \quad z_i \leq 1 \text{ for } i \in I\}.$$
Note that $Z$ is defined by the linear constraints (3a) and (3b) together with the continuous relaxation of the binary conditions. A suitable measure for the violation of the complementarity conditions (3c) is simply given by $z^\top (q + Mz)$, which is non-negative on $Z$. On the other hand, the violation of the binary conditions (3d) can be measured in different ways. Several penalty functions have been proposed in the literature; see, e.g., De Santis et al. (2013), Giannessi and Tardella (1998), Lucidi and Rinaldi (2010), Rinaldi (2009), and Zhu (2003). For our purposes, a concave and piecewise linear penalty function is preferred. Thus, we choose the classic penalty function
\[
\sum_{i \in I} \min\{ z_i, 1 - z_i \}
\]
and obtain the following nonlinear, nonconvex, and nonsmooth reformulation
\[
\min_{z \in Z} f(z) := \alpha z^\top (q + Mz) + (1 - \alpha) \sum_{i \in I} \min\{ z_i, 1 - z_i \}.
\]
Here, $\alpha \in (0, 1)$ is a parameter controlling the emphasis that is put on each of the two penalty terms. Note that if (3) is solvable, Problem (6) is an equivalent reformulation, i.e., its global solutions are solutions of (3)—and vice versa. Otherwise, if (3) is not solvable, the global solutions of problem (6) are approximate solutions of (3) with the smallest violation of the complementarity and the integrality conditions. Here, the violation of both the complementarity as well as the integrality conditions are measured in the $\ell_1$ norm. It is our aim now to devise a penalty branch-and-bound method to solve Problem (6) that exploits the specific structure of the objective function $f$ in (6).

In classic branch-and-bound approaches for mixed-integer problems, branching is done by starting with the continuous relaxation and by creating different subproblems in which variables, which are fractional in the relaxation’s solution, are fixed to certain values or the feasible set is divided into disjoint sets using inequalities. Global upper bounds are derived by feasible points and local lower bounds are obtained from solving the optimization problems in the nodes of the branch-and-bound tree. In our method, however, we start with an $\alpha$-scaled version of the continuous relaxation (2) and create subproblems by adding different penalty terms (for fractional variables) to the objective function. Thus, in contrast to classic branch-and-bound methods, the feasible set remains the same over the entire tree. The bounding is done in analogy to a classic branch-and-bound as the objective values of the node problems yield local lower bounds and any feasible solution yields a global upper bound.

3.1. Branching. At the root node of the branch-and-bound tree, we solve the convex relaxation
\[
\min_{z \in Z} \alpha z^\top (q + Mz)
\]
of Problem (6) that is obtained by neglecting the second term in the objective function. Afterward, two child nodes are created as follows. Each node corresponds to a new convex QP, in which we add either $(1 - \alpha)z_j$ or $(1 - \alpha)(1 - z_j)$ to the objective function. To this end, we choose an index $j \in I$ that satisfies $\min\{z_j^*, 1 - z_j^*\} > 0$ in the solution $z^*$ of the root node relaxation. We will discuss more sophisticated branching rules in Section 4.

In particular, the problem of the first child node reads
\[
\min_{z \in Z} f_1(z) := \alpha z^\top (q + Mz) + (1 - \alpha)z_j,
\]
which aims to get $z_j$ close to 0 in the respective subtree, while the problem of the second child node is given by
\[
\min_{z \in Z} f_2(z) := \alpha z^\top (q + Mz) + (1 - \alpha)(1 - z_j),
\]
which aims to get $z_j$ close to 1 in the respective subtree. The idea is to split the term $\min\{z_j, 1 - z_j\}$ occurring in (6) into two new problems and taking the minimum of both these problems, i.e.,

$$\min_{z \in Z} \alpha z^T(q + Mz) + (1 - \alpha) \min\{z_j, 1 - z_j\} = \min_{z \in Z} \left\{ \min_{z \in Z} f_1(z), \min_{z \in Z} f_2(z) \right\}$$

Note that both $z_j$ and $1 - z_j$ are non-negative on the feasible set $Z$. At an arbitrary node of the branch-and-bound tree, we thus have a convex-quadratic problem in which non-negative terms $(1 - \alpha)z_j$ and $(1 - \alpha)(1 - z_j)$, $j, k \in I$, have been added to the convex-quadratic function $\alpha z^T(q + Mz)$. We define $I_0$ to be the set of indices $j \in I$ for which $(1 - \alpha)z_j$ has been added and $I_1$ to be the set of indices $j \in I$ for which $(1 - \alpha)(1 - z_j)$ has been added. This definition is in close analogy to the sets of fixed variables in a classic branch-and-bound method for mixed-binary problems. Consequently, every node $N$ is uniquely determined by $N = (I_0, I_1)$. In the following, the objective function at a node $N = (I_0, I_1)$ is denoted by

$$f_N(z) = \alpha z^T(q + Mz) + (1 - \alpha) \left( \sum_{j \in I_0} z_j + \sum_{j \in I_1} (1 - z_j) \right)$$

and the optimal solution of that node is denoted by $z_N^*$. Thus, the problem that has to be addressed at a node $N = (I_0, I_1)$ is of the form

$$\min_{z \in Z} f_N(z).$$

(7)

Without loss of generality, we assume that the problems in the first child nodes are always obtained adding the terms $(1 - \alpha)z_j$, $j \in I_0$, which we call “downwards branching”. The problems in the second child nodes are obtained adding the terms $(1 - \alpha)(1 - z_j)$, $j \in I_1$, which we call “upwards branching”.

As a first result, we show that enumerating all possible partitions $(I_0, I_1)$ of $I$, i.e., $I = I_0 \cup I_1$ with $I_0 \cap I_1 = \emptyset$, yield the optimal value of Problem (6). In other words, we show that the minimum among the optimal solutions of the problems of all leaf nodes of the fully enumerated branch-and-bound tree is the optimal solution of Problem (6).

**Lemma 1.** Let $z^*$ be an optimal solution of Problem (6). Then, it holds

$$f(z^*) = \min \{ f_N(z_N^*) : N = (I_0, I_1) \text{ with } I_0 \cup I_1 = I \text{ and } I_0 \cap I_1 = \emptyset \}.$$ 

**Proof.** Note that the feasible set does not change from one $N$ to another. Hence, all optimal points are feasible for all nodes. Let $N^* = (I_0^*, I_1^*)$ be the leaf with $I_0^* := \{ i \in I : z_i^* \leq 1 - z_i^* \}$ and $I_1^* := \{ i \in I : z_i^* > 1 - z_i^* \}$. We then have

$$f(z^*) = \alpha(z^*)^T(q + Mz^*) + (1 - \alpha) \sum_{j \in I} \min \{ z_j^*, 1 - z_j^* \}$$

$$= \alpha(z^*)^T(q + Mz^*) + (1 - \alpha) \sum_{j \in I_0^*} z_j^* + (1 - \alpha) \sum_{j \in I_1^*} (1 - z_j^*)$$

$$= f_{N^*}(z_{N^*}^*) \geq f_{N^*}(z_{N^*}^*).$$

Hence,

$$f(z^*) \geq \min \{ f_N(z_N^*) : N = (I_0, I_1) \text{ with } I_0 \cup I_1 = I \text{ and } I_0 \cap I_1 = \emptyset \}$$

holds. To show the other inequality, we assume that there exists a node $N' = (I_0', I_1')$ with $I_0' \cup I_1' = I$ and $I_0' \cap I_1' = \emptyset$ such that

$$f_{N'}(z_{N'}^*) < f(z^*)$$
holds. We thus obtain \( f_N(z^*_N) < f(z^*_N) \) or, equivalently,
\[
\alpha(z^*_N)^\top(q + Mz^*_N) + (1 - \alpha) \sum_{j \in I_0^l} z^*_N,j + (1 - \alpha) \sum_{j \in I_1^l} (1 - z^*_N,j) \\
< \alpha(z^*_N)^\top(q + Mz^*_N) + (1 - \alpha) \sum_{j \in I} \min \{z^*_N,j, 1 - z^*_N,j\}.
\]
This implies
\[
\sum_{j \in I_0^l} (z^*_N,j - \min \{z^*_N,j, 1 - z^*_N,j\}) + \sum_{j \in I_1^l} (1 - z^*_N,j - \min \{z^*_N,j, 1 - z^*_N,j\}) < 0,
\]
which is impossible as
\[
z^*_N,j \geq \min \{z^*_N,j, 1 - z^*_N,j\}
\]
and
\[
1 - z^*_N,j \geq \min \{z^*_N,j, 1 - z^*_N,j\}.
\]
Hence,
\[
f(z^*) \leq \min \{f_N(z^*_N) : N = (I_0, I_1) \text{ with } I_0 \cup I_1 = I \text{ and } I_0 \cap I_1 = \emptyset\}
\]
holds and the claim follows. \( \square \)

As a consequence of this lemma, we know that by iterating over all possible partitions of \( I \), we get an optimal solution of Problem (6), which is key to prove the correctness of the overall penalty branch-and-bound method.

3.2. Bounding. Similar to classic branch-and-bound methods, we can establish local lower bounds on the optimal solution for the different nodes and global upper bounds for the optimal solution of Problem (6). Obviously, the value of \( f \) at any feasible point is a global upper bound. Hence, \( f(z^*) \leq f(z^*_N) \) holds with \( N \) being an arbitrary node of the branch-and-bound tree. We denote by \( z^*_\text{inc} \) the incumbent, i.e., the point so that \( f(z^*_\text{inc}) \) constitutes the best known upper bound for Problem (6) found so far.

Next we prove that the optimal value of the problem defined at a certain node is a lower bound for the optimal value of the problem defined at any of its successor nodes. While this is rather trivial for classic branch-and-bound methods, this result is harder to establish for the penalty branch-and-bound method considered here.

**Lemma 2.** Let \( N' = (I_0', I_1') \) be a successor of some node \( N = (I_0, I_1) \) in the branch-and-bound tree, i.e., \( I_0 \subseteq I_0' \) and \( I_1 \subseteq I_1' \) holds. Then,
\[
f_N(z^*_N) \leq f_{N'}(z^*_N')
\]
holds.

**Proof.** Since the feasible set does not change during the branching process all feasible points remain feasible for all nodes. Thus,
\[
f_{N'}(z^*_N') = \alpha(z^*_N')^\top(q + Mz^*_N') + (1 - \alpha) \sum_{j \in I_0'} z^*_N',j + (1 - \alpha) \sum_{j \in I_1'} (1 - z^*_N',j) \\
= \alpha(z^*_N)^\top(q + Mz^*_N) + (1 - \alpha) \sum_{j \in I_0} z^*_N,j + (1 - \alpha) \sum_{j \in I_1} (1 - z^*_N,j) \\
+ (1 - \alpha) \sum_{j \in I_0' \setminus I_0} z^*_N',j + (1 - \alpha) \sum_{j \in I_1' \setminus I_1} (1 - z^*_N',j) \\
\geq \alpha(z^*_N)^\top(q + Mz^*_N) + (1 - \alpha) \sum_{j \in I_0} z^*_N,j + (1 - \alpha) \sum_{j \in I_1} (1 - z^*_N,j) \\
= f_N(z^*_N) \geq f_N(z^*_N).
\]
Note that the first inequality is due to the fact that $z^*_N,j \geq 0$ and $(1 - z^*_N,j) \geq 0$ for $j \in I$ on the feasible set. The second inequality follows from optimality.

Lemma 2 implies that in the case that Problem (7) at node $N$ leads to a solution $z^*_N$ such that

$$f_N(z^*_N) \geq f(z^*_{inc})$$

holds, we have that every leaf of the subtree rooted in $N$ cannot yield a better solution than the best known solution $z^*_{inc}$. Hence, we can prune the subtree rooted in $N$. Note that in our branch-and-bound method, there is no direct analogy to pruning due to feasibility as done in classic branch-and-bound methods: As soon as we find a feasible solution for Problem (3) we stop the algorithm. Pruning because of infeasibility is also not possible here, since the feasible set does not change throughout the process.

3.3. The Algorithm. We are now ready to formally state the basic scheme of our penalty branch-and-bound method in Algorithm 1.

Algorithm 1 A Penalty Branch-and-Bound Algorithm for MILCPs

Input: $q \in \mathbb{R}^n$, $M \in \mathbb{R}^{n \times n}$, $\emptyset \subseteq I \subseteq [n]$, $\alpha \in (0,1)$

Output: A global optimum $z^*$ of Problem (6).

Set $N \leftarrow \{(\emptyset, \emptyset)\}$, $f_{inc} \leftarrow \infty$, and $z^*_{inc} \leftarrow \text{none}$.

while $N \neq \emptyset$ do

Choose $N = (I_0, I_1) \in N$ and set $N \leftarrow N \setminus \{N\}$.

Compute $z^*_N \in \arg \min \{f_N(z) : z \in Z\}$.

if $f(z^*_N) < f_{inc}$ then

Set $z^*_{inc} \leftarrow z^*_N$ and $f_{inc} \leftarrow f(z^*_N)$.

if $f_N(z^*_N) < f_{inc}$ and $I \setminus (I_0 \cup I_1) \neq \emptyset$ then

Choose $j \in I \setminus (I_0 \cup I_1)$, set $N \leftarrow N \cup \{(I_0 \cup \{j\}, I_1), (I_0, I_1 \cup \{j\})\}$.

return $z^*_{inc}$

Theorem 1. Algorithm 1 terminates after finitely many steps with a global optimal solution of Problem (6).

Proof. The algorithm terminates after finitely many steps since the set $I$ is finite. Thus, at some point, $I = I_0 \cup I_1$ holds and we can no longer find a branching variable in the node and no child node can be generated. Assume now that $f_N(z^*_N) < f(z^*_{inc})$ always holds in the second if-clause. Then the correctness of the algorithm follows from Lemma 1, as we iterate through the complete branch-and-bound tree. Finally, in the cases, in which $f_N(z^*_N) \geq f(z^*_{inc})$ holds, the nodes that are not added can be excluded due to Lemma 2.

4. Further Algorithmic Enhancements

Similar to classic branch-and-bound methods, there are different possibilities to improve the performance of the overall algorithm. Both the choice of the next node to be solved and the choice of the next index to branch on are not yet specified in Algorithm 1—although it is known that these aspects have a significant impact on the performance of the algorithm. We address these choices in Sections 4.1 and 4.2. In Section 4.3, we also briefly discuss the possibility of warmstarting the problem of each node with the optimal basis of the parent node. One of the most important ingredients of classic branch-and-bound algorithms are additional valid inequalities that can be incorporated in the problems of the nodes to improve the dual bound. We investigate two different types of valid inequalities for MILCPs in Section 4.4.
4.1. Node Selection. In our implementation of the penalty branch-and-bound algorithm, we consider three different node selection strategies. The first two are depth- and breadth-first search. The third strategy is based on Lemma 2 and will be referred to as the “lower-bound-push strategy”.

From Lemma 2, we know that the optimal value $f_N(z_N^*)$ of the problem defined at a node $N$ is a local lower bound for the subtree rooted in $N$. Hence, the global lower bound is the smallest value among the lower bounds obtained from nodes that have unsolved children. As the node to be solved next, we thus select the child of the node $N$ that has the lowest objective value $f_N(z_N^*)$. When both children of $N$ are not yet solved, we evaluate the objective function of both child nodes at the optimal solution of the parent node. Then, we choose the child node with the smaller value as we would expect this to result in a smaller lower bound. This lower bound may then be improved in the new node.

In our numerical experiments, we consider depth- and breadth-first search strategies as a benchmark for the lower-bound-push strategy; see Section 5.

4.2. Branching Rules. It is known that the performance of branch-and-bound methods for mixed-integer problems strongly depends on the branching strategy (Achterberg et al. 2005), i.e., on how to select the next variable to branch on.

In every node of our penalty branch-and-bound method, we need to choose an index $j \in I$ to define the objective functions of the problems in the child nodes. Clearly, we do not branch on a variable $z_j$, $j \in I$, if this variable is integer-valued in this node.

In the following, we propose two different branching strategies: “pseudocost branching”, which is well-known from mixed-integer programming and “MIQP-based branching”. In our numerical experiments, we compare these two strategies with random branching (i.e., the naive approach of choosing the index $j \in I$ at random) and most-violated branching (i.e., choosing the index of the variable closest to 1/2).

4.2.1. Pseudocost Branching. As a first advanced approach, we consider a variant of pseudocost branching—a technique commonly used in branch-and-bound algorithms for mixed-integer programs that goes back to Benichou et al. (1971). The idea of pseudocost branching is to measure the expected objective gain when branching on a specific variable index. The strategy is to keep track of the change in the objective function when an index $j \in I$ has been chosen to be branched on. The rule then chooses the index that is predicted to have the largest impact on the objective function based on these past changes.

We transfer this idea to our context in the following. Let $\varphi^1_{N,j}$ be the objective gain per unit change when branching upwards on variable $j \in I$ at node $N$:

$$\varphi^1_{N,j} := \frac{f(z_{N,j}^*) - f(z_N^*)}{|z_{N,j}^*| - z_{N,j}^*}.$$ 

Here, $N_j$ is the child of $N$ created by upwards branching. We denote by $\psi^1_j$ the expected objective gain per unit change when branching upwards on variable $j$. To this end, let $N^j$ be the set of nodes where $j \in I$ is chosen as the variable to branch on. Then, we define $\psi^1_j$ as

$$\psi^1_j := \frac{1}{|N^j|} \sum_{N^j} \varphi^1_{N,j}.$$ 

Analogously, we define $\varphi^0_{N,j}$ and $\psi^0_j$ for branching downwards on variable $j \in I$.

The average gain is then calculated as

$$s_j := \mu \min \left\{ \psi^0_j \cdot (z_{N,j}^* - |z_{N,j}^*|), \psi^1_j \cdot ([z_{N,j}^*] - z_{N,j}^*) \right\} + (1 - \mu) \max \left\{ \psi^0_j \cdot (z_{N,j}^* - |z_{N,j}^*|), \psi^1_j \cdot ([z_{N,j}^*] - z_{N,j}^*) \right\}$$

where $\mu$ is a parameter to be determined.
with \( \mu \in (0, 1) \). The pseudocost-based branching candidate then is the index \( j \in I \) with the largest score \( s_j \). At the beginning of our branch-and-bound, we initialize the average \( \psi_j^{0.1} \) with 1. If at a certain node \( N \), we have not yet branched on a candidate \( j \in I \), namely \( N^j = \emptyset \), we initialize that \( \psi_j^{0.1} \) with the average of all other \( \psi_i^{0.1} \) for \( i \in I \) with \( i \neq j \).

4.2.2. MIQP-Based Branching. As a further branching rule that we use in the penalty branch-and-bound, we propose a strategy based on solving a single-variable MIQP for each integer variable in the presolve phase of the algorithm. Again, we aim at sorting the indices \( j \in I \) so that we branch on those indices first that are expected to give good lower bounds on the optimal solution. For every index \( j \in I \), we solve the following MIQP with a single integer variable:

\[
\begin{align*}
\min_{z \in \mathbb{R}^n} & \quad z^\top (q + Mz) \\
\text{s.t.} & \quad q + Mz \geq 0, \quad z \geq 0, \\
& \quad z_j \in \{0, 1\}.
\end{align*}
\]

As discussed in the introduction, we know that it is likely that the overall MILCP has no solution and that this is due to the combination of complementarity as well as integrality conditions. By solving all \( |I| \) many MIQPs (8) we measure the impact of the \( i \)th binary variable on the infeasibility of the problem (if it is infeasible at all). The indices \( j \in I \) are then sorted with decreasing optimal objective function values of Problem (8). Moreover, infeasible problems are formally assigned the objective function value \( \infty \). The resulting branching strategy then chooses the branching candidate on top of the list while skipping all integer-feasible indices as well as all indices that have been branched on already.

4.3. Warmsstarting. Recall that the feasible set stays the same over the entire search tree and that the objective functions change only slightly from a parent node to its child nodes. This allows for warmsstarting the QP solver for solving the child nodes. To this end, we take the optimal primal basis of the parent node as the starting basis for the child nodes. Note that this only works if we do not add further valid inequalities since these can render the old basis infeasible for the child problem. Thus, we need to numerically evaluate later on if warmsstarting or valid inequalities have the greater effect on the overall performance of the method.

4.4. Valid Inequalities. In this section, we propose two classes of inequalities that are not valid for the overall problem (6) in the classic sense but that are valid locally.

4.4.1. Simple Cuts. The first class of inequalities are called simple cuts. Assume that we just solved node \( N \) and that we decide to branch on the variable \( z_j, j \in I \). Then, in the nodes corresponding to the downwards branching subtree, we add the bound constraint \( z_j \leq 0.5 \). While in the nodes belonging to the upwards branching subtree, we add the bound constraint \( z_j \geq 0.5 \). Although this sounds rather simple, the effect of including these cuts is significant (see Section 5) and proving the correctness of these inequalities is also not as easy as stating them.

We first show that the optimal solution of Problem (6) is not cut off when introducing the simple cuts. To this end, we prove that the minimum among the optimal solutions of the leaf problems that we obtain when including the simple cuts still is the optimal solution of Problem (6).

**Lemma 3.** Let 

\[ z_N^\star \in \arg \min \{ f_N(z) : z \in Z, z_{I_0} \leq 0.5, z_{I_1} \geq 0.5 \} \]
be an optimal solution at node $N$ when simple cuts are included. Then,
\[
f(z^*) = \min \{ f_N(z_N^*) : N = (I_0, I_1) \text{ with } I_0 \cup I_1 = I \}
\]
holds.

**Proof.** Let $N^* = (I_0^*, I_1^*)$ be the leaf defined by $I_0^* := \{ j \in I : z_j^* \leq 1 - z_j^* \}$ and $I_1^* := \{ j \in I : z_j^* > 1 - z_j^* \}$. We then have
\[
f(z^*) = \alpha(z^*) \top (q + M z^*) + (1 - \alpha) \sum_{i \in I} \min \{ z_i^*, 1 - z_i^* \}
\]
\[
= \alpha(z^*) \top (q + M z^*) + (1 - \alpha) \sum_{j \in I_0^*} z_j^* + (1 - \alpha) \sum_{j \in I_1^*} (1 - z_j^*)
\]
\[
= f_{N^*}(z^*) \geq f_{N^*}(z_{N^*}^*).
\]
The last inequality holds because, by definition, we have $z_j^* \leq 0.5$ for all $j \in I_0^*$ and $z_j^* \geq 0.5$ for all $j \in I_1^*$. Thus, $z^*$ is feasible for $N = (I_0^*, I_1^*)$, which is a leaf by definition. Hence,
\[
f(z^*) \geq \min \{ f_N(z_N^*) : N = (I_0, I_1) \text{ with } I_0 \cup I_1 = I \}
\]
holds. To show the other inequality, we assume that there exists a node $N' = (I_0', I_1')$ with $I_0' \cup I_1' = I$ such that
\[
f_{N'}(z_{N'}^*) < f(z^*)
\]
holds. With $f(z^*) \leq f(z_{N'}^*)$, we obtain
\[
f_{N'}(z_{N'}^*) < f(z_{N'}^*)
\]
or, equivalently,
\[
\alpha(z_{N'}^*) \top (q + M z_{N'}^*) + (1 - \alpha) \sum_{j \in I_0'} z_{N',j}^* + (1 - \alpha) \sum_{j \in I_1'} z_{N',j}^*
\]
\[
< \alpha(z_{N'}^*) \top (q + M z_{N'}^*) + (1 - \alpha) \sum_{j \in I} \min \{ z_{N',j}^*, 1 - z_{N',j}^* \}.
\]
This implies
\[
\sum_{j \in I_0'} \left( z_{N',j}^* - \min \{ z_{N',j}^*, 1 - z_{N',j}^* \} \right) + \sum_{j \in I_1'} \left( 1 - z_{N',j}^* - \min \{ z_{N',j}^*, 1 - z_{N',j}^* \} \right) < 0,
\]
which is a contradiction by definition. Hence,
\[
f(z^*) \leq \min \{ f_N(z_N^*) : N = (I_0, I_1) \text{ with } I_0 \cup I_1 = I \}
\]
holds and the claim follows.  

As a second result, we show that Lemma 2 is also valid when simple cuts are used in the branch-and-bound method.

**Lemma 4.** Let $N' = (I_0', I_1')$ be a successor of some node $N = (I_0, I_1)$ in the branching tree, i.e., $I_0' \subseteq I_0'$ and $I_1' \subseteq I_1'$ holds. Further, let $z_N^*, z_{N'}^*$ be optimal solutions of nodes $N$ and $N'$, respectively, when simple cuts are used. Then,
\[
f_N(z_N^*) \leq f_{N'}(z_{N'}^*)
\]
holds.
Proof. By definition, we have
\[
 f_N(z_N^*) = \alpha(z_N^*)^\top (q + Mz_N^*) + (1 - \alpha) \sum_{j \in I_0} z_{N,j}^* + (1 - \alpha) \sum_{j \in I_1} (1 - z_{N,j}^*) \\
 = \alpha(z_N^*)^\top (q + Mz_N^*) + (1 - \alpha) \sum_{j \in I_0} z_{N,j}^* + (1 - \alpha) \sum_{j \in I_1} (1 - z_{N,j}^*) \\
 + (1 - \alpha) \sum_{j \in I_0 \setminus I_1} z_{N,j}^* + (1 - \alpha) \sum_{j \in I_1 \setminus I_0} (1 - z_{N,j}^*) \\
 \geq \alpha(z_N^*)^\top (q + Mz_N^*) + (1 - \alpha) \sum_{j \in I_0} z_{N,j}^* + (1 - \alpha) \sum_{j \in I_1} (1 - z_{N,j}^*) \\
 = f_N(z_N^*) \geq f_N(z_N^*).
\]
Note that the first inequality holds since \( z_{N,j}^* \geq 0 \) and \( (1 - z_{N,j}^*) \geq 0 \) are valid on the feasible set. The last inequality holds because the feasible sets of the nodes are nested in the sense, that the feasible set of node \( N' \) is a subset of the feasible set of node \( N \). Hence, every feasible point of \( N' \) is also feasible for \( N \). □

Theorem 2. Algorithm 1 remains correct when simple cuts
\[ z_j \leq 0.5 \text{ for all } j \in I_0, \quad z_j \geq 0.5 \text{ for all } j \in I_1 \]
are added at any node \( N = (I_0, I_1) \).

Proof. From Lemma 3, we know that the optimal solution of Problem (6) is the optimal solution of a leaf node. From Lemma 4, we know that the objective value of every ancestor node of a leaf yields a lower bound for the objective value of this leaf. Hence, if we have a feasible point \( z_{\text{inc}} \) of Problem (6) and some node \( N \) for which
\[ f(z_{\text{inc}}^*) \leq f_N(z_N^*) \]
holds, we know that \( z_{\text{inc}}^* \) is a solution that is as good as every solution that any leaf being a successor of \( N \) can yield. Thus, we can prune the subtree rooted in \( N \). The same applies for the case in which a node problem becomes infeasible due to the introduction of cuts. Hence, Algorithm 1 remains correct when simple cuts are used. □

4.4.2. Optimality Cuts. The second class of inequalities that we introduce are so-called optimality cuts. In order to define them, we use the necessary optimality conditions for Problem (6); see, e.g., Corollary 3.68 in Beck (2017). Let \( z^* \in Z \) be an optimal solution of Problem (6), then \( g \in \partial f(z^*) \) exists such that
\[ g^\top (z - z^*) \geq 0 \text{ for all } z \in Z. \]
Hence, if we find a point \( z^* \) during our branch-and-bound search that does not fulfill this inequality for any known feasible point \( z \in Z \), we can cut off \( z^* \). In particular, we derive the valid inequality
\[ g^\top z' \geq g^\top z, \]
with \( z' \in Z \) being some fixed feasible solution. Furthermore, for any \( \bar{g}, \tilde{g} \in \partial f(z) \) such that \( \tilde{g}^\top z' \geq g^\top z' \) and \( \bar{g}^\top z \leq g^\top z \) holds, the following inequality is also valid:
\[ \tilde{g}^\top z' \geq \bar{g}^\top z. \]
This will be necessary to convexify the valid inequality.
Lemma 5. Let \( z' \in Z \) be a feasible solution and let \( N = (I_0, I_1) \). Then, \[ \alpha z^\top (q + 2Mz) + (1 - \alpha) \sum_{j \in I_0} z_j + (1 - \alpha) \sum_{j \in I_1} (1 - z_j) - (1 - \alpha)|I \setminus I_0| \leq \alpha(z')^\top (q + 2Mz) + (1 - \alpha) \sum_{j \in I_\setminus I_1} z'_j \]
is a valid inequality for the subtree rooted at node \( N \).

Proof. Let \( z' \in Z \), \( z \in Z \), and \( g \in \partial f(z) \) be given. We need to underestimate \( g^\top z \) and overestimate \( g^\top z' \). The \( i \)th component of \( g \in \partial f(z) \) is given by

\[ g_i = \alpha q_i + \alpha \sum_{j \in [\alpha]} 2M_{i,j}z_j \begin{cases} + (1 - \alpha), & \text{for } z_i < 0.5, \ i \in I, \\ - (1 - \alpha), & \text{for } z_i > 0.5, \ i \in I, \\ + (1 - \alpha)y_i, & \text{for } z_i = 0.5, \ i \in I, \\ 0, & \text{for } i \not\in I, \end{cases} \]

for some \( y_i \in [-1, 1] \).

We can then underestimate \( g^\top z \) as follows:

\[ g^\top z = \alpha z^\top (q + 2Mz) + (1 - \alpha) \left( \sum_{i \in I, z_i < 0.5} z_i - \sum_{i \in I, z_i > 0.5} z_i + \sum_{i \in I, z_i = 0.5} y_i^2 z_i \right) \]

\[ \geq \alpha z^\top (q + 2Mz) + (1 - \alpha) \left( \sum_{i \in I, z_i < 0.5} z_i - \sum_{i \in I, z_i > 0.5} z_i - \sum_{i \in I, z_i = 0.5} z_i \right) \]

\[ = \alpha z^\top (q + 2Mz) + (1 - \alpha) \left( \sum_{i \in I, z_i < 0.5} z_i + \sum_{i \in I, z_i > 0.5} (1 - z_i) - \sum_{i \in I, z_i = 0.5} 1 \right) \]

\[ \geq \alpha z^\top (q + 2Mz) + (1 - \alpha) \sum_{i \in I} \min\{z_i, 1 - z_i\} - (1 - \alpha)|I| \]

\[ \geq \alpha z^\top (q + 2Mz) + (1 - \alpha) \sum_{i \in I_0} z_i + (1 - \alpha) \sum_{i \in I_1} (1 - z_i) - (1 - \alpha)|I|. \]

Note that the term \(|I|\) can be replaced by \(|I \setminus I_0|\) if simple cuts are included. On the other hand, we can overestimate \( g^\top z' \) as follows:

\[ g^\top z' = \alpha(z')^\top (q + 2Mz) + (1 - \alpha) \left( \sum_{i \in I, z_i < 0.5} z'_i - \sum_{i \in I, z_i > 0.5} z'_i + \sum_{i \in I, z_i = 0.5} y_i^2 z'_i \right) \]

\[ \leq \alpha(z')^\top (q + 2Mz) + (1 - \alpha) \left( \sum_{i \in I, z_i < 0.5} z'_i - \sum_{i \in I, z_i > 0.5} z'_i + \sum_{i \in I, z_i = 0.5} z'_i \right) \]

\[ = \alpha(z')^\top (q + 2Mz) + (1 - \alpha) \left( \sum_{i \in I, z_i < 0.5} z'_i - \sum_{i \in I, z_i > 0.5} z'_i \right) \]

\[ \leq \alpha(z')^\top (q + 2Mz) + (1 - \alpha) \sum_{i \in I_\setminus I_1} z'_i - (1 - \alpha) \sum_{i \in I_1, z_i \neq 0.5} z'_i \]
\[ \leq \alpha(z')^\top (q + 2MZ) + (1 - \alpha) \sum_{i \in I \setminus I_1} z'. \]

The combination of the two inequalities yields the lemma. \(\Box\)

5. Numerical Results

We start with describing the software and hardware setup of our numerical tests and discuss the test set. We implemented the penalty branch-and-bound method presented in Section 3 in Python 3.7. All node problems are solved with the QP solver of Gurobi 9.0.3 and all the tests were run on an Intel Xeon CPU E5-2699 v4 @ 2.20GHz (88 cores) with 756 GB RAM. In this section, we refer to the implementation of Algorithm 1 as MILCP-PBB. For our tests, we consider instances that we randomly generated as follows. The matrices \(M \in \mathbb{R}^{n \times n}\) have been created using the sprandsym function of MATLAB for sizes \(n \in \{50, 100, 150, 200, 250, 300, 350, 400, 450, 500\}\). Details on the spectra and the densities of the matrices can be found in Appendix A. We then built vectors \(q \in \mathbb{R}^n\) in four different ways, each reflecting a certain “degree of feasibility” in the resulting instance. Let \(z^* \in \mathbb{R}^n\) be a solution of an instance of Problem (3). Then, it satisfies

1. Feasibility w.r.t. \(Z\): \(z^* \in Z\),
2. Integrality: \(z_i^* \in \{0, 1\}\) for all \(i \in I\),
3. Complementarity: \((z^*)^\top (q + Mz^*) = 0\).

The vectors \(q\) have been created to satisfy at least one of the conditions above. More precisely, we built instances for which \(z \in \mathbb{R}^n\) exists so that

(a) only Condition (i) is guaranteed to be satisfied,
(b) only Conditions (i) and (ii) are guaranteed to be satisfied,
(c) only Conditions (i) and (iii) are guaranteed to be satisfied,
(d) all Conditions (i)–(iii) are guaranteed to be satisfied.

We created 10 instances for every size \(n\) and the types (a)–(c), yielding 300 different instances in total. Type (d) appeared to be very easy to solve, which is why we exclude these instances from the test set. Again, more details on how the test set has been built can be found in Appendix A.

For the comparisons presented in this section we use logarithmic performance profiles in the sense of Dolan and Moré (2002).

In Section 4, we discussed a number of ways to enhance Algorithm 1. To determine the best setting for MILCP-PBB, we study (i) the impact of different branching rules, (ii) different node selection strategies, (iii) the use of warmstarts, and (iv) the use of valid inequalities. This is done in Sections 5.1–5.4, respectively, where we set a time limit of 1 h. In Section 5.5, the best parameterization of MILCP-PBB is then compared with two other approaches. The first one is proposed in Gabriel, Conejo, Ruiz, et al. (2013), where the authors reformulate the MILCP as an MILP using additional binary variables and big-\(M\) constraints to re-model the complementarity constraints. The second approach is inspired by this model but uses an MIQP instead of an MILP reformulation of the given MILCP.

5.1. Comparison of Different Branching Rules. We now compare the performance of MILCP-PBB when equipped with the four different branching rules described in Section 4.2. For these tests, the node selection strategy is set to breadth-first search, warmstarts are disabled, and no valid inequalities are added. For the pseudocost branching strategy, we set \(\mu = 0.5\). We exclude 11 instances from the test set since no parameterization is able to solve them within the time limit. Figure 1 displays the performance profiles w.r.t. the running time and the number of branch-and-bound nodes. One can see that the running time and the number of nodes for the random branching rule, the pseudocost branching strategy,
and the branching strategy based on the most fractional variable do not differ much. However, the MIQP-based branching rule yields a significant improvement both in terms of the number of nodes and the running time. This improvement is especially true for the number of nodes, as our MIQP-based approach visits significantly fewer nodes for the vast majority of the instances, while also solving the overall largest number of instances to global optimality. The improvement regarding the running times is not as significant. This is to be expected since the ordering of branching priorities during the presolve phase is more expensive compared to the computational effort required by the other branching strategies. However, the advantage regarding the number of nodes overcompensates this disadvantage and the MIQP-based branching rule also dominates all other strategies w.r.t. running times as well.

5.2. Comparison of Different Node Selection Strategies. We now compare the three node selection strategies described in Section 4.1. To this end, we use the MIQP-based branching strategy, while warmstarts and valid inequalities are disabled. We exclude 12 instances from the set since no parameterization of our method is able to solve them within the time limit. Based on Figure 2, one can notice that the node selection strategies only have a minor impact on the performance of the overall method both in terms of the number of nodes and the running time. The lower-bound-push strategy seems to have a slight advantage regarding the required number of branch-and-bound nodes, but also has a slight disadvantage regarding the running time. Again, this is due to the higher computational cost for the ordering of the nodes. It seems that the depth-first search strategy is a bit faster than the breadth-first search strategy. However, the latter solves slightly more instances, which is why we choose it for our “best-setting” implementation of MILCP-PBB.

5.3. The Benefits of Warmstarts. We now compare the performance of MILCP-PBB with and without warmstarts. To this end, we use the MIQP-based approach branching rule, the breadth-first search node selection strategy, and avoid the use of any valid inequalities. In case that warmstarts are used, we need to solve the node problems using the primal simplex method within Gurobi. However, this leads to some numerical instabilities during our preliminary testing. Thus, we implemented a backup strategy that disables warmstarts in the case of numerical troubles and then allows that Gurobi chooses any other method for solving the node problems.
We exclude 49 instances from the set as no parameterization is able to solve them within the time limit. As expected, warmstarts significantly help to reduce the running time; see Figure 3 (right). Let us finally comment on the surprising result that using warmstarts or not leads to a different number of branch-and-bound nodes required to solve the problems; see Figure 3 (left). This is due to the occurrence of node problems with non-unique optimal solutions. In such a case, using warmstarts or not might lead to different solutions of the node problems, which, in turn, affects the overall search tree.

5.4. Computational Analysis of the Valid Inequalities. We tested different types of valid inequalities as described in Section 4.4. Unfortunately, incorporating the optimality cuts (see Section 4.4.2) results in severe numerical troubles for Gurobi. Possible reasons for that might be that these cuts are both quadratic second-order-cone constraints and very dense. We also tried different relaxations of these cuts to obtain sparser cuts but this did not resolve the numerical troubles. We also tested a linearized version of this quadratic cut. This resolved almost all numerical issues but, on the other hand, lead to the fact that we do not cut off any points anymore. Thus,
making these optimality cuts work in a practical implementation is still subject to future work. Consequently, we only consider simple cuts. Note that these cuts can be set up at no computational cost and that it is to be expected that they only have a minimal impact on the computational time required to solve the node problems since they are merely variable bounds. We compare a version of MILCP-PBB in which all possible simple cuts are added in every node with a version of MILCP-PBB in which no simple cuts are added. For this test, the branching rule is set to the MIQP-based branching rule, the node selection strategy is set to breadth-first search, and warmstarts are disabled. Note, that warmstarts and valid inequalities are mutually exclusive since warmstarts require the primal simplex method for solving the node problems but the primal basis of a parent node is not feasible anymore after adding a valid inequality in a child node. No instances are excluded for this test. As can be seen in Figure 4, incorporating the simple cuts has a great impact both on the number of branch-and-bound nodes as well as on the running time.

5.5. Comparing MILCP-PBB with Other Approaches. Our preliminary numerical tests reveal that the best parameterization of MILCP-PBB uses the MIQP-based branching rule and adds all possible simple cuts at every node. As mentioned before, the use of warmstarts and the usage of simple cuts are mutually exclusive. Hence, we decide to disable warmstarts because the simple cuts have a bigger positive impact on the overall performance of MILCP-PBB. As mentioned before, we choose breadth-first search as our node selection strategy.

In order to compare MILCP-PBB with other approaches from the literature, we consider what is proposed in Gabriel, Conejo, Ruiz, et al. (2013). There, the authors reformulate the MILCP problem as an MILP with additional binary variables and big-M constraints to re-model complementarity constraints. Note that we use the notation $B$ for the large constants to avoid confusion with the LCP’s matrix $M$. 

![Figure 4. Performance profiles for the number of branch-and-bound nodes (left) and the running time (right) for variants with all possible simple cuts and without any.](image-url)
The respective MILP then reads as follows:

\[
\begin{align*}
\min_{z, z', z'', \rho, \sigma} \quad & \quad \alpha \sum_{i=1}^{n} \rho_i + (1 - \alpha) \sum_{i \in I} \sigma_i, \\
\text{s.t.} \quad & \quad z \geq 0, \quad q + Mz \geq 0, \\
& \quad z \leq B z' + \rho, \\
& \quad q + Mz \leq B(1 - z') + \rho, \\
& \quad 0 \leq z_I \leq z'' + \sigma, \\
& \quad z'' - \sigma \leq z_I \leq 1, \\
& \quad z \in \mathbb{R}^n, \quad z' \in \{0,1\}^n, \quad z'' \in \{0,1\}^I, \\
& \quad \sigma \in \mathbb{R}^I_0, \quad \rho \in \mathbb{R}^n_0.
\end{align*}
\] (9a) (9b) (9c) (9d) (9e) (9f) (9g) (9h)

Similar to Formulation (5), \( \rho_i \) bounds the violation of each complementarity constraints, while \( \sigma_i \) bounds the violation of the binary constraints. The variables \( z'_i \) are indicator variables that decide if for index \( i \) the corresponding variable \( z_i \) or \((q + Mz)_i\) is as close as possible to 0. Analogously, the indicator variables \( z''_i, i \in I \), decide if the corresponding variable \( z_i \) is as close as possible to 0 or to 1. Note that this formulation will result in different optimal objective function values compared to our approach as the violation of the complementarity constraint is penalized in a different way. Furthermore note that Model (9) requires a significantly larger set of \( 3n + 2|I| \) variables.

Besides the significantly larger number of variables, one additional drawback of Model (9) is that it requires to determine sufficiently large big-B constraints. However, we can actually modify Problem (9) to get rid of these big-Bs and to measure the violation of the complementarity constraints using the same term as in our approach. This comes at the price of considering a quadratic instead of a linear problem. This resulting MIQP is given by

\[
\begin{align*}
\min_{z, z', \sigma} \quad & \quad \alpha z^T(q + Mz) + (1 - \alpha) \sum_{i \in I} \sigma_i, \\
\text{s.t.} \quad & \quad z \geq 0, \quad q + Mz \geq 0, \\
& \quad 0 \leq z_I \leq z' + \sigma, \\
& \quad z'' - \sigma \leq z_I \leq 1, \\
& \quad z \in \mathbb{R}^n, \quad z' \in \{0,1\}^n, \quad z'' \in \{0,1\}^I, \\
& \quad \sigma \in \mathbb{R}^I_0.
\end{align*}
\] (10a) (10b) (10c) (10d) (10e) (10f)

Instead of using variables \( \rho_i \) for bounding the violation of the complementarity, we use the direct penalization via the corresponding quadratic term. The violation of the binary constraints is still measured in the same way as in the MILP (9), with \( z'_i \) being the corresponding indicator variables as before.

Note that the MIQP (10) only has \(|I|\) additional binary variables \( z' \) and \(|I|\) additional continuous variables \( \sigma_i \) when compared to the original MILCP. Thus, the number of additionally required auxiliary variables is significantly reduced compared to the MILP reformulation (9). This makes a huge difference in practice: Gurobi is able to solve the MIQP (10) in significantly less time compared to what is required for solving the MILP (9); see Figure 5. When (9) and (10) are solved using Gurobi, all presolve techniques and heuristics have been disabled. For obtaining a fair comparison, we further restrict both the MIQP solver of Gurobi and the QP solver of Gurobi used for solving the nodes within MILCP-PBB to only use a single thread.

For a first comparison we use the same instances as before. Figure 5 shows the
performance profiles of MILCP-PBB and Gurobi for both the MILP and the MIQP formulation w.r.t. the number of nodes and running time. It can be seen that MILCP-PBB needs significantly fewer nodes, while still needing more running time. The increased running time can probably be attributed to time inefficiencies from our Python implementation. As the MIQP reformulation has no failures and MILCP-PBB only has five, we increased the difficulty of the test set to have a further comparison on a harder test set for the MIQP reformulation and MILCP-PBB. As the other two methods clearly outperform the MILP formulation, we do not compare the MILP formulation on the more difficult set. We built a second test set of 300 random instances as before but double both the instance sizes and the integer densities. For this test, we also tripled the time limit and now consider as failures those instances that are not solved within 3 h.

The comparison of the methods applied to these instances is shown in Figure 6, where we excluded 97 instances since no method is able to solve them. We can notice that, again, the number of nodes needed by MILCP-PBB is significantly smaller than the one needed by Gurobi. Gurobi is, in general, faster but has significantly more unsolved instances and MILCP-PBB turns out to be more robust. MILCP-PBB and Gurobi has 26 and 48, respectively, failures on instances of Type (a), 50 and 60 failures on instances of Type (b), as well as 33 and 43 failures on instances of Type (c). A possible explanation for these results is the difference in the size of the respective branching trees. As the size of a branch-and-bound tree grows exponentially with the number of binary variables, the larger number of nodes in the tree of the MIQP reformulation becomes even larger for the more difficult instances. While Gurobi needs less time per node and probably also finds the optimal solution, the sheer size of the tree disallows proving optimality within the time limit.

6. Conclusion

We presented, analyzed, enhanced, and tested a novel penalty branch-and-bound method for solving MILCPs. Here, “solving” means that we indeed compute a solution if one exists or that we compute an approximate solution that minimizes an infeasibility measure based on the violation of the integrality and complementarity conditions of the problem. Together with further MILCP-tailored enhancements such as valid inequalities, node selection strategies, branching rules, and warmstarting.
techniques, this leads to a method that significantly outperforms two benchmark approaches that we compared our method with.

Despite these contributions, many interesting research questions remain open from which we finally want to sketch three:

1. Is it possible to extend our penalty branch-and-bound method to MILCPs with indefinite matrices $M$? In this case, the QPs to be solved in the nodes of our branch-and-bound tree become nonconvex so that one either needs to exploit global optimization techniques in these nodes or to modify the bounding step applied during the search of the tree.

2. Are there further techniques as used in classic branch-and-bound for MILPs such as presolve or heuristics as well as additional valid inequalities, etc. that are explicitly tailored to the setting of MILCPs? In particular, the question remains open on how the optimality cuts can be modified and implemented so that they are useful for practical computations.

3. Finally, it would be interesting to see specific applications of MILCPs solved with our method. Moreover, a further interesting aspect would be the problem-specific development of techniques such as valid inequalities etc. to additionally speed up the solution process.

ACKNOWLEDGMENTS

The first author acknowledges support within the project RM120172A2970290, which has received funding from Sapienza, University of Rome. The third author thanks the DFG for their support within project A05 and B08 in the “SFB TRR 154 Mathematical Modelling, Simulation and Optimization using the Example of Gas Networks”. The last author has been supported by the German Research Foundation (DFG) within the Research Training Group 2126: “Algorithmic Optimization”.

APPENDIX A. DETAILED DESCRIPTION OF THE TEST SET

In order to build a proper test set of MILCP instances, we created matrices $M \in \mathbb{R}^{n \times n}$ using the sprandym function of MATLAB for sizes $n \in \{50, 100, 150, 200, 250, 300, 350, 400, 450, 500\}$. The corresponding matrix densities
have been chosen so that they roughly follow the sigmoid-like function

\[ d(n) := \frac{1}{1 + e^{\frac{n}{n-5}}} \]

Moreover, we obtain a random non-negative spectrum with an upper bound of 100 for the eigenvalues.

The set of integral variables \( I \) has been chosen as a random sample of size

\[ r(n) := \frac{1}{5 \left( 1 + e^{\frac{3n-3}{n}} \right)} \]

Finally, we built vectors \( q \in \mathbb{R}^n \) for the four different “degrees of feasibility”; see Section 5. In order to build instances of Type (a), for which only feasibility with respect to \( Z \) is guaranteed (i.e., Condition (i) is satisfied), we set \( q = x - Mz \) starting from two random vectors \( x, z \in \mathbb{R}^n \) such that \( x \geq 0, z \geq 0, \) and \( z_I \in [0,1]^I \). Note that it is possible that this process yields instances for which the integrality or complementarity constraints are satisfied as well—although this is rather unlikely. Instances of Type (b), for which feasibility with respect to \( Z \) (Condition (i)) and integrality (Condition (ii)) are guaranteed, have been built by setting \( q = x - Mz \) with \( x, z \in \mathbb{R}^n \) being randomly generated so that, besides \( x \geq 0 \) and \( z \geq 0, \) also \( z_I \in \{0,1\}^I \) holds. In order to build instances of Type (c), for which feasibility w.r.t. \( Z \) (Condition (i)) and the complementarity constraint (Condition (iii)) are fulfilled, we set \( q = -Mz \) with \( z \in \mathbb{R}^n \) being a randomly created point with \( z \geq 0 \) and \( z_I \in [0,1]^I \). Note that this is the same procedure as for the first test set. Instances of Type (d), for which all three conditions are fulfilled, have been built by setting \( q = -Mz \) with \( z \in \mathbb{R}^n \) being a randomly created point with \( z \geq 0 \) and \( z_I \in \{0,1\}^I \) (as we did for the instances of Type (b)). The “degree of feasibility” of the instance clearly has a significant impact on its difficulty; see Figure 7, where a comparison of the performances of MILCP-PBB with different branching rules on the instances is reported.

Instances of Type (d) that have been created to be feasible both for the complementarity as well as the integrality conditions, turned out to be very easy. Most of them have been solved in the root node of the corresponding branch-and-bound tree. Thus, we decided to exclude them from our computational analysis. Instances of Type (a) and (b) that not have been forced to be feasible w.r.t. the complementarity conditions and which are either forced to be integer-feasible or not are also solved rather quickly. The most complicated instances are those of Type (c) which are forced to be feasible w.r.t. the complementarity conditions but which are not forced to be integer feasible. This is possibly related to the difference in which the violation of the complementarity constraint and the violation of the integrality constraints are penalized along the nodes of MILCP-PBB. While the term penalizing the violation of the complementarity constraint is added to every node problem, we are penalizing the violation of all integrality constraints only at the leaf nodes. Hence, the lower bound for instances that are complementarity feasible but that are not forced to be integer feasible will stay closer to zero for longer.

Due to these preliminary tests and experiments, we decided to construct matrices with 5% density and we further adjusted the integer densities to make the instances of Type (a)–(c) comparably difficult. For instances of Type (a), the integer density is thus set to 8%, for the instances of Type (b), the integer density is set to 4%, and for instances of Type (c), we set the integer density to 10%.

References

Figure 7. Test of different branching rules in dependence of the “degree of feasibility” of the instance
REFERENCES


(M. de Santis) Sapienza Università di Roma, Department of Computer, Control, and Management Engineering, Via Ariosto 25, 00185 Roma, Italy
Email address: marianna.desantis@uniroma1.it

(S. de Vries, M. Schmidt, L. Winkel) Trier University, Department of Mathematics, Universitätsring 15, 54296 Trier, Germany
Email address: devries@uni-trier.de
Email address: martin.schmidt@uni-trier.de
Email address: winkell@uni-trier.de