A Penalty Branch-and-Bound Method for Piecewise Convex Objective Functions

Vom Fachbereich IV der Universität Trier zur Verleihung des akademischen Grades eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte

Dissertation

betreut und begutachtet von
Prof. Dr. Sven de Vries und Prof. Dr. Martin Schmidt

vorgelegt von

Lukas Winkel

Trier, 2023

Berichterstatter:	Prof. Dr. Sven de Vries
	Prof. Dr. Martin Schmidt

Curriculum Vitae

04/2019 – heute	**Mitglied im Graduiertenkolleg ALOP** Prof. Dr. Volker Schulz
03/2019 11/2016 – 03/2019	**Master of Science, Wirtschaftsmathematik** Studium der Wirtschaftsmathematik an der Universität zu Köln
11/2016 10/2013 – 11/2016	**Bachelor of Science, Wirtschaftsmathematik** Studium der Wirtschaftsmathematik an der Universität zu Köln
03/2013 08/2004 – 07/2013	**Abitur** Quirinus-Gymnasium, Neuss
08/2000 – 06/2004	**Gemeinschaftsgrundschule** Hemmerden, Grevenbroich

This thesis is dedicated to my family and my friends, who stuck with me through the ups and downs that are inherent to the dissertation process, who supported me through the hardships of my illness and kept me sane through the challenges of the global pandemic, who proofread, who listened to me complain, who understood me without understanding. I want to thank my supervisors, who helped me, not only while healthy, but especially while my health prevented me from working on finishing this thesis. I could not have asked for a better support system.

Thank you!

“Humans are not good at maths. People see people who are good at maths and think: ≫They must be geniuses, it must come naturally to them.≪ For the vast majority of people who are into maths, they don’t find it easy. They are just people who enjoy, how difficult it is.”

– Matt Parker (2019)

“99 % of mathematics is not understanding stuff and 1 % is wondering why you didn’t understand it earlier.”

– Unknown

Zusammenfassung

Die Dissertation beschäftigt sich mit einer neuartigen Art von Branch-and-Bound Algorithmen, deren Unterschied zu klassischen Branch-and-Bound Algorithmen darin besteht, dass das Branching durch die Addition von nicht-negativen Straftermen zur Zielfunktion erfolgt anstatt durch das Hinzufügen weiterer Nebenbedingungen. Die Arbeit zeigt die theoretische Korrektheit des Algorithmusprinzips für verschiedene allgemeine Klassen von Problemen und evaluiert die Methode für verschiedene konkrete Problemklassen. Für diese Problemklassen, genauer Monotone und Nicht-Monotone Gemischtganzzahlige Lineare Komplementaritätsprobleme und Gemischtganzzahlige Lineare Probleme, präsentiert die Arbeit verschiedene problemspezifische Verbesserungsmöglichkeiten und evaluiert diese numerisch. Weiterhin vergleicht die Arbeit die neue Methode mit verschiedenen Benchmark-Methoden mit größtenteils guten Ergebnissen und gibt einen Ausblick auf weitere Anwendungsgebiete und zu beantwortende Forschungsfragen.

Contents

Chapter 1

Introduction

The strive for optimization is older than humankind itself. Since the beginning of time, nature optimized itself through the means of evolution. Living creatures changed and optimized their behavior and physical characteristics in order to gain an advantage. But also inanimate objects optimize certain features naturally. Bubbles form close to perfect spheres in order to minimize the surface to volume ratio and water flows or light rays take the shortest path through a medium. Humans of all millennia further developed these undetermined and randomly happening optimization processes by making deliberate decisions in order to minimize travel times or effort and maximize benefits and profits. Today, society in its modern form would be unimaginable without the optimization efforts of the past. GPS navigation systems tell us the shortest way to the airport, airlines schedule their work force in an optimized way, manufacturers design their airplanes to improve flight performance, investors invest in these companies based on optimization methods and advertisements for our holidays are shown to people most likely to book it. Today, every major decision all around us is supported by some form of optimization to improve results. The rise of artificial intelligence in the 21$^{\text{st}}$ century subsequently put optimization techniques in the spotlight of a broader audience although under a slightly different name. In every case, the goal is to find the "best" solution out of the possibilities available.

The earliest formalizations of optimization problems in the mathematical sense are from the 17$^{\text{th}}$ and 18$^{\text{th}}$ century, where mathematicians such as Pierre de Fermat, Joseph-Louis Lagrange, Isaac Newton, and Carl Friedrich Gauss came up with first methods to find mathematically optimal solutions. With the beginning of the 20$^{\text{th}}$ century efforts to categorize and solve optimization problems increased and formed *Mathematical Optimization* into its own subfield of mathematics. In the mathematical sense, an optimization problem consists of two components. One is the set of constraints, that describes the set of all possibilities, the so-called feasible set. The second is the objective function, i.e., a quantitative measure of quality for every element of the feasible set. Then, the goal of optimizing is to find the element in the feasible set, which has the lowest or highest objective value. A mathematical optimization problem can be formalized in the following way.

Definition 1.0.1 (General Optimization Problem)**.** Let $X \subseteq \mathbb{R}^n$ be the feasible set and $f : X \mapsto \mathbb{R}$ be the objective function. The corresponding optimization problem reads

$$\min_{x \in \mathbb{R}^n} \quad f(x) \tag{1.0.1a}$$
$$\text{s.t.} \quad x \in X. \tag{1.0.1b}$$

In this general form, the problems can be arbitrarily difficult depending on the properties of the objective function and the feasible set. Hence, there is no one-solves-all method for this general form. Therefore, mathematicians began categorizing the problems regarding different features. A first big differentiation is the one into unconstrained problems, i.e., problems with a feasible set $X = \mathbb{R}^n$, and constrained problems, i.e., problems with a feasible set $X \subsetneq \mathbb{R}^n$. For a long time constrained problems were mostly categorized into linear, i.e., problems where both the objective function is linear and the feasible set can be described by linear inequalities, and nonlinear optimization problems. For linear problems (LP), many fast solution methods exist and, in general, solving LPs is not a challenge. But as a famous quote of unknown origin says:

"Categorizing mathematical objects into linear and nonlinear is like categorizing real objects into banana and non-banana.",

which is why for the last decades a new, more important classification became dominant. Nowadays, the most important distinction is the one between convex optimization problems, i.e., problems where both the objective function and the feasible set are convex, and non-convex problems. This distinction appears to be a good measure for the difficulty of problems, as a lot of optimization theory, that allows for faster solution methods, only holds for convex problems. For example, the first-order necessary optimality conditions first stated by William Karush in 1939 as reported by Kuhn (1976) are sufficient for convex optimization problems and locally optimal points are also globally optimal. Linear problems are trivially convex problems and therefore unsurprisingly fall into the easier-to-solve category. For non-convex problems it is oftentimes too difficult to find global optima and one has to settle for the search of local optima instead of global ones, or additional measures to convexify the problem or to exploit locally convex structures have to be taken. An example for such a technique are *Spatial Branch-and-Bound* methods (Liberti et al. (2006)). There is one exception for the research into non-convex optimization problems. Technically, integrality constraints are non-convex constraints, but, because they have a nice combinatorial structure and have been well researched for decades, a lot of good solution methods exist and (mixed-)integer constraints are considered individually, even though theoretical complexity remains the same. For other non-convexities there still are not many global solution techniques.

This thesis ventures into the realms of global non-convex optimization and presents a novel global solution technique for certain non-convex and non-smooth optimization problems, that have a convex feasible set and a non-convex objective function with a certain combinatorial structure. We present a novel type of branch-and-bound algorithm, which is able to solve different optimization problems with non-convex and non-smooth objective functions. We present the algorithm for different possible problem classes, investigate further enhancement strategies, and test it numerically.

1.1. State-of-the-Art

In this section, we will give a brief introduction into the two known techniques, that the main algorithmic concept of this thesis is based on. In Section 1.1.1, we will introduce the general principles of branch-and-bound (BB) methods, which go back to Land and Doig (1960). The second technique is the reformulation of problems via the means of penalty functions, where some constraints of the problem are replaced by penalty terms in the objective function, which we will discuss in Section 1.1.2. The method we present later on is a fusion of both approaches, where a novel branch-and-bound method is used to solve penalty reformulations of the investigated problem classes.

1.1.1. Branch-and-Bound

Branch-and-bound methods are a class of divide-and-conquer algorithms that have been developed in order to solve (mixed-)integer optimization problems. We will present the principles of this algorithm class on the example of mixed-integer linear problems (MILP), which are defined in the following. It has to be noted that MILP are only decidable in general, if the integer variables are bounded by the constraint set (Benichou et al. (1971)). As the class of bounded and integer constrained LPs is equivalent to the class of binary constrained LPs, we will concern ourselves predominantly with binary constrained problems, but we will use the terms “integer constraints” and “binary constraints” interchangeably.

Definition 1.1.1 (Mixed-Integer Linear Problem)**.** Let $c \in \mathbb{R}^n,\ b \in \mathbb{R}^m$ be vectors, $A \in \mathbb{R}^{m\times n}$ a matrix and $I \subseteq \{1, \ldots, n\}$ a set of indices. Then, the corresponding Mixed-Integer Linear Problem is:

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & c^\top x && \text{(1.1.1a)} \\ \text{s.t.} \quad & Ax \le b, && \text{(1.1.1b)} \\ & x \ge 0, && \text{(1.1.1c)} \\ & x_i \in \{0,1\} \text{ for all } i \in I. && \text{(1.1.1d)} \end{aligned}$$

Without the integer constraints, it would be easy to solve the resulting LP. The branch-and-bound algorithm therefore removes all integrality constraints at first and then adds them in successively. By doing this, the feasible set is divided into subsets to be investigated individually. This is the *branching* part of the method. During the process, some subsets of the feasible set are disregarded, when it can be proven that they do not contain an optimal point. This is the *bounding* part of the method. We will look into more details in the following sections.

Branching

As mentioned before, all integer constraints are removed at first and the resulting linear relaxation is solved.

Definition 1.1.2 (Linear Relaxation of a MILP)**.** The linear relaxation of the MILP defined in Definition 1.1.1 is given by

$$
\begin{aligned}
\min_{x \in \mathbb{R}^n} \quad & c^\top x && \text{(1.1.2a)}\\
\text{s.t.} \quad & Ax \leq b, && \text{(1.1.2b)}\\
& x \geq 0, && \text{(1.1.2c)}\\
& x_i \in [0, 1] \text{ for all } i \in I. && \text{(1.1.2d)}
\end{aligned}
$$

This will be the problem in the root node of the so-called branch-and-bound tree, which is built from here on out.

In order to build the tree, the optimal solution of the root node is investigated. If no integrality constraints are violated, we have found the optimal solution of the MILP. If there is a variable, which is supposed to be integer but is fractional, there are two possibilities for the optimal solution of the MILP. That variable has to be either 0 or 1. Two child problems are created for both possibilities. We will denote one of the indices for which the integrality constraint is violated by $j \in I$. The two child nodes can then be written in the following way.

Definition 1.1.3 (Left Child of the Root Node of BB for MILP)**.** The left child of the root node problem defined in Definition 1.1.2 is defined as

$$
\begin{aligned}
\min_{x \in \mathbb{R}^n} \quad & c^\top x && \text{(1.1.3a)}\\
\text{s.t.} \quad & Ax \leq b, && \text{(1.1.3b)}\\
& x \geq 0, && \text{(1.1.3c)}\\
& x_i \in [0, 1] \text{ for all } i \in I, && \text{(1.1.3d)}\\
& x_j = 0. && \text{(1.1.3e)}
\end{aligned}
$$

Definition 1.1.4 (Right Child of the Root Node of BB for MILP)**.** The right child of the root node problem defined in Definition 1.1.2 is defined as

$$
\begin{aligned}
\min_{x \in \mathbb{R}^n} \quad & c^\top x && \text{(1.1.4a)}\\
\text{s.t.} \quad & Ax \leq b, && \text{(1.1.4b)}\\
& x \geq 0, && \text{(1.1.4c)}\\
& x_i \in [0, 1] \text{ for all } i \in I, && \text{(1.1.4d)}\\
& x_j = 1. && \text{(1.1.4e)}
\end{aligned}
$$

Both problems are then solved independently of each other and the process is repeated. Optimal solutions of the node problems are evaluated and new child nodes are created for an index for which integrality constraints are violated. This results in a binary tree structure as every node with violated integrality constraints will get two child nodes. Every node in that tree has a certain structure, that can be defined by the indices and binary values, that were used in the branching process up to that point in the tree. We will use the following notation. A node $N = (I_0, I_1)$ is defined by two sets I_0 and I_1. The set I_0 contains all branching decisions that were taken along the branch from the node N to the root node, where the ancestor of N was the left child. In other words, $j \in I_0$ denotes the fact, that at some point along the bpath from root node to that node, the constraint $x_j = 0$ was added to the constraint set. The set I_1 is defined analogously for right child nodes.

Definition 1.1.5 (Node Problem of BB for MILP)**.** Let $N = (I_0, I_1)$ be a node in the tree. Then, the node problem reads

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & c^\top x && (1.1.5a)\\ \text{s.t.} \quad & Ax \leq b, && (1.1.5b)\\ & x \geq 0, && (1.1.5c)\\ & x_i \in [0,1] \text{ for all } i \in I, && (1.1.5d)\\ & x_j = 0 \text{ for all } j \in I_0, && (1.1.5e)\\ & x_j = 1 \text{ for all } j \in I_1. && (1.1.5f) \end{aligned}$$

Note, that we obtain the root node problem for $I_0 = I_1 = \emptyset$. It can be shown, that by taking the optimal solution of a leaf node problem, i.e., a problem of a node that has no children of its own, that minimizes the objective function over all optimal solutions of leaf node problems, we get the optimal solution of the MILP. While this procedure would solve the MILP, it would take a lot of time as the number of nodes would grow exponentially with the number of integer variables. We therefore want to take measures to avoid enumerating all possible leaf node problems.

Bounding

In order to avoid enumerating and evaluating all possible subproblems created by the branching part of the algorithm, we want to use upper and lower bounds on the optimal objective value. These bounds can be established during the branching process in every node of the tree. As a first step it is important to notice that the feasible set of the root node problem is a superset of the feasible set of the original MILP and the feasible sets of all other node problems. Therefore, the optimal objective value of the root node problem is a lower bound of the optimal objective value of the original problem and all child node problems. Furthermore, when we consider the tree to be a directed tree with edge directions going away from the root node towards the leaves of the tree, the feasible sets along a directed path of the tree are nested in the sense that the feasible set of a node problem is a subset of the feasible set of its parent node problem. Hence, the optimal objective value of every node problem is a local lower bound for every node problem in the subtree rooted in that node.

Secondly, every integer feasible solution yields a global upper bound, as the optimal integer feasible solution has to be at least as good as that. Integer feasible solutions either appear in the branching process by chance if an optimal solution of a node problem is integer feasible or they can be computed by heuristics. The best known integer feasible point is called the incumbent and will be denoted by x^*_{inc}.

From these bounds, three possibilities arise to dispose a node problem in the tree and stop adding child nodes to that node. This is called *pruning*. The first possibility is pruning because of suboptimality. If the optimal objective value of a node problem is larger than the best upper bound known for x^*_{inc}, the optimal solution of the MILP cannot appear in the subtree rooted in that node, because we already have a point that is at least as good as the best possible point that could come up. Therefore, no new child nodes have to be added. The second possibility is pruning due to integer feasibility. If the optimal solution of a node problem is integer feasible, we can show that that point is already the best solution possible in that subtree and therefore the subtree does not have to be investigated further. The third possibility is pruning because of infeasibility. As the feasible set gets smaller along a branch of the tree, it is possible that a node problem becomes infeasible and hence trivially the subtree rooted in that node cannot contain the optimal solution of the MILP. In practice, this allows for educed branch-and-bound trees and faster solution times, even though the worst-case complexity is still exponential.

Algorithmic Description

We now want to bring the branching and bounding together and give the algorithmic description of the method. For now, we denote the feasible set of a node Problem N defined by the tuple (I_0, I_1) by X_N.

Algorithm 1 A Branch-and-Bound Algorithm for MILPs

Input: $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{m\times n}$, $I \subseteq \{1, \dots, n\}$.
Output: A global optimum x^* of Problem 1.1.1 or indication of infeasibility.
Set $\mathcal{N} \leftarrow \{(\emptyset, \emptyset)\}$, $f_{\text{inc}} \leftarrow \infty$, and $x^*_{\text{inc}} \leftarrow$ none.
while $\mathcal{N} \neq \emptyset$ **do**
  Choose $N = (I_0, I_1) \in \mathcal{N}$ and set $\mathcal{N} \leftarrow \mathcal{N} \setminus \{N\}$.
  Compute $x^*_N \in \operatorname{argmin}\{c^\top x : x \in X_N\}$.
  if x^*_N exists
    if $c^\top x^*_N < f_{\text{inc}}$ **and** $(x^*_N)_I \in \{0,1\}^I$ **do**
      Set $x^*_{\text{inc}} \leftarrow x^*_N$ and $f_{\text{inc}} \leftarrow c^\top x^*_N$.
    if there is a $j \in I$ with $(x^*_N)_j \notin \{0,1\}$ **do**
      Choose $j \in I$ with $(x^*_N)_j \notin \{0,1\}$.
      Set $\mathcal{N} \leftarrow \mathcal{N} \cup \{(I_0 \cup \{j\}, I_1), (I_0, I_1 \cup \{j\})\}$.
return x^*_{inc}

Theorem 1.1.6. *If there are upper and lower bounds on all integer variables and the root node problem is bounded, Algorithm 1 terminates after finitely many steps with a global optimal solution of Problem 1.1.1.*

A proof of this theorem can be found in many publications on branch-and-bound methods, such as the original paper by Land and Doig (1960) and will therefore be omitted at this point.

Further Enhancements

So far we have stated the general principles of branch-and-bound methods, but there are further enhancements possible. As mentioned before, fast and problem specific heuristics can be used to improve the upper bound $c^\top x^*_{\text{inc}}$ early on. Also, in the algorithmic description we have not specified on how we choose both the next subproblem to solve $N \in \mathcal{N}$ and the next index $j \in I$ to branch on. While the choice is irrelevant for the correctness of the method, it may have a big impact on the performance as the order of choices impacts the quality of both the upper and the lower bound and there are often many candidates to choose from. Investigations into both choices are for example done in Achterberg et al. (2005) and Wojtaszek and Chinneck (2010). Another technique to increase performance is warmstarting the node problems. As we progress down in the tree, the node problems along a branch only change slightly, which is why we can expect that in many cases the optimal solution of a node problem only differs by little to the optimal solutions of its child nodes. Therefore, it can be sensible to use the optimal basis vector of a node as the starting basis for the simplex method used to solve the child nodes. In alot of cases, problems can also be simplified before starting the progress using pre-solve techniques. Aside from these possibilities, usually the biggest impact on the performance have valid inequalities. These are inequalities, that can be added to the constraint set of a node and are supposed to cut off fractional optimal solutions of the relaxations solved, while not cutting of the optimal solution of the MILP. By doing so, the relaxation is tightened. Such cuts have been investigated extensively and many different types of valid inequalities have been found, examples can be found in Cornuéjols (2008).

1.1.2. Penalty Formulations

The second concept we briefly want to introduce are penalty reformulations and methods. Here, some or all constraints are omitted from the constraint set and added as penalty terms to the objective function to model the constraints. For minimization problems, a penalty term for a constraint is a function, that is strictly non-negative for any point that violates the constraint and zero otherwise. In other words, if we have a constraint $h(x) \geq 0$, a penalty term for that constraint is a function f with

$$f(x) \begin{cases} > 0, & \text{if } h(x) < 0, \\ = 0, & \text{else.} \end{cases}$$

Usually, the value of the penalty term increases with an increase of some measure of violation of the constraint. Therefore, if we add a penalty term to the objective function the optimal solution should tend to a lower violation. By including penalty terms instead of constraints, constrained optimization problems can be converted into unconstrained problems. For example, a way to model an equality constraint $h(x) = 0$ is adding a quadratic penalty term $h(x)^2$, as the function $h^2(x)^2$ has a minimum in $h(x) = 0$, which is also the point for which the constraint would not be violated. Details on penalty methods can be found in Nocedal and Wright (1999). For other types of constraints, there are more specific penalty functions. Penalty functions for binary constraints have been investigated in many publications such as in De Santis et al. (2013), Giannessi and Tardella (1998), Lucidi and Rinaldi (2010), Rinaldi (2009), and Zhu (2003). They can be modeled by basically any function that is strictly non-negative on the interval $[0, 1]$ and zero if and only if $x_j \in \{0, 1\}$. One way to penalize non-integrality is linearly. Here, the farther the variable x_j is from both 0 and 1, i.e., the closer it is to 0.5, the higher is the penalization, while growing linearly. This penalty function can be described by a minimum function in the following way: $\min\{x_i, 1 - x_i\}$. The penalization of multiple integrality constraints for an index set I with the sum of such minimum functions, i.e.,

$$P_I(x) := \sum_{i \in I} \min\{x_j, 1 - x_j\},$$

will have a prominent role in subsequent chapters.

However, there are some things to consider. Usually, by moving constraints to the objective function as penalty terms, the objective function becomes rather complicated. Also, oftentimes, the penalty reformulation is not exact and only approximates the optimal solution of the underlying problem.

Feasibility Problems

In the case of feasibility problems, i.e., problems that do not have an objective function, showing that a penalty reformulation is exact is easy. We can solve the feasibility problem by solving the reformulation to global optimality, because whenever the objective value of the reformulation is 0 for a point, this point is feasible for the original problem. Further, if the optimal objective value of the reformulation is larger than zero, the original problem does not have a feasible point. Additionally, in case there is no point that is actually feasible, the penalty reformulation has the advantage of delivering a point that minimizes the violation of the constraints that were moved to the objective function, which might be reasonable in some applications. So if we have, for example, the following feasibility problem

$$\begin{aligned}
\exists \quad & x \in \mathbb{R}^n && \text{(1.1.6a)}\\
\text{s.t.} \quad & h_i(x) = 0 \text{ for all } i \in J_E, && \text{(1.1.6b)}\\
& h_i(x) \geq 0 \text{ for all } i \in J_I, && \text{(1.1.6c)}
\end{aligned}$$

we can, for example, move all equality constraints to the objective function in the following way

$$\min_{x \in \mathbb{R}^n} \quad \sum_{i \in J_E} \mu_i h_i^2(x) \tag{1.1.7a}$$

$$h_i(x) \geq 0 \text{ for all } i \in J_I, \tag{1.1.7b}$$

and solve the reformulation to solve the feasibility problem. In that case, the vector $\mu \in \mathbb{R}^{|J_E|}_{>0}$ is a vector of emphasis parameters. With that vector, we can stress the importance of each constraint. In case there is no point that is feasible for all constraints, higher parameter values for some constraints prioritize these constraints over others. For solving the original feasibility problem, the values of the parameters do not matter as the globally optimal point of the reformulation delivers a feasible point for all parameter vectors, if there is one.

Optimality Problems

When the original problem already has a non-trivial objective function, it becomes more difficult. Now the value of the emphasis parameters matters for the exactness of the reformulation. If the parameters are too small, the penalization might be too little and the globally optimal point of the reformulation might not be feasible for the original problem. If the parameters are too big, numerical troubles might arise. Also, it is non-trivial to decide, how big a parameter has to be so that the reformulation is exact finds the optimal solution of the original problem. A way to tackle this challenge is to start with smaller parameters and increase them if the computed solution is infeasible. It can be shown that there is a parameter big enough, but similarly to other big-M formulations, it is in general not easy to say, when the method can stop increasing the parameter. Also, repeatedly solving the problem is computationally expensive.

1.2. Contributions

In this thesis, a novel branch-and-bound method is presented. For the method to be applicable, the feasible set needs to be convex and the objective function needs to have a certain combinatorial structure but does not need to be convex. In classic branch-and-bound methods, the branching is done by successively adding constraints to relaxations of the original problem while the objective function stays the same. Our method works differently. We exploit the non-convex and non-smooth structure of the objective function of the problem and realize the branching decisions by successively adding non-negative terms to a relaxed version of the original problem. Different to classic branch-and-bound methods, the feasible set remains the same throughout the process. Problems, that have the structure needed to apply our method often arise in penalty reformulations. For example, the penalty term P_I introduced in Section 1.1.2 has such a structure. We propose the method for different reformulations of problem classes and numerically compare them to other benchmark approaches with mostly competitive results. Furthermore, we investigate which problem classes could be solved theoretically by such a method and characterize two very general possible problem classes. These problem classes have a non-convex, non-smooth but piecewise convex objective function in common.

1.3. Structure and Notation

The remainder of the thesis is structured in the following way. In Chapter 2 we present the problem class of Mixed-Integer Linear Complementarity Problems, for which we originally developed the method and explain the principles of our novel branch-and-bound method using this problem class. We present possible problem specific enhancements and compare our method with a benchmark approach. In Chapter 3 we describe two general possible problem classes, that can be theoretically solved by the method as a proof of concept. In Chapter 4, we present a fourth problem class, which is the canonical extension of the problem class from Chapter 2 and for which we investigate and improve our method. Again, we test the improved version of the algorithm against a benchmark approach. In Chapter 5 we present possible use cases of the method for the broad class of MILP. Here, we not only investigate solving MILPs with the method but make use of the implicit ability of the algorithm to find lower and upper bounds on the optimal objective value. We conclude the thesis in Chapter 6.

There will be recurring notation we briefly want to introduce now. In the optimization problems that follow, the vector of optimization variables is referred to as x or z. The feasible sets are referred to as X or Z and the objective functions are referred to as f. During branch-and-bound processes, the objective function of a specific node N is referred to as f_N. Usually, x^* or z^* refers to the optimal solution of an optimization problem. Other functions are mostly denoted by g or h and refer to specific terms in the objective function or the constraint set. The index set I refers to the set of indices of the binary variables of the problems. A vector x_I then refers to the subvector of x containing all variables with indices in I. During the branching process, the index sets I_0 and I_1 refer to the indices for which branching decisions were made. Further, $[n]$ refers to the range of integers from 1 to n, i.e., $[n] := \{1, \ldots, n\}$.

1.4. Acknowledgement

The author has been supported by the German Research Foundation (DFG) within the Research Training Group 2126: “Algorithmic Optimization”.

Chapter 2

A Novel Approach to Monotone Mixed-Integer Linear Complementarity Problems

In this chapter we introduce our novel branch-and-bound method using the class of problems for which we originally devised the method for as an example. In Section 2.1 we introduce said problem class and the state-of-the-art research on these problems. In Section 2.2 we present the basic principles of the algorithm and prove its correctness. In Section 2.3 we investigate possible problem specific enhancements of the method, which we will test numerically in Section 2.4. Here, we also compare our method to benchmark reformulations based on a solution approach from the literature. Parts of this chapter and the results have already been published in Santis et al. (2022).

2.1. The Linear Complementarity Problem

Linear complementarity problems (LCP) are a well studied class of feasibility problems. They consist of linear inequality and quadratic equality constraints and are generally non-convex. They have been studied since the middle of the 20^{th} century and have many applications in different fields. In mathematics itself, the KKT conditions for some optimization problems, e.g., quadratic problems, can be modeled as LCPs. In related fields, such as mechanics, economics and game theory, many equilibrium problems can be modeled and analyzed by reformulating them as LCPs. An extensive overview over the state-of-the-art on LCPs can be found in the seminal textbook by Cottle et al. (2009).

2.1.1. The Continuous Case

The continuous version of an LCP is defined as follows:

Definition 2.1.1. Let $q \in \mathbb{R}^n$, $M \in \mathbb{R}^{n \times n}$ be given. The linear complementarity problem denoted by LCP(q, M) is the task to find a vector $z \in \mathbb{R}^n$ that satisfies

$$z \geq 0, \tag{2.1.1a}$$
$$q + Mz \geq 0, \tag{2.1.1b}$$
$$z^\top(q + Mz) = 0 \tag{2.1.1c}$$

or to show that no such vector exists.

For continuous LCPs the two biggest questions are the questions of existence and uniqueness of solutions. Many theoretical results and characterizations of matrix classes, for which the corresponding LCPs have certain properties, exist for these questions. There are also many reformulations and algorithms to solve LCPs constructively and find a solution. For example, there is Lemke's Algorithm, a simplex-like method based on a basis exchange principle, which is described in Lemke and Howson (1962). Another possibility is the following quadratic problem (QP), which is a penalty reformulation:

Reformulation 2.1.2 (QP Penalty Reformulation of an LCP)**.** The LCP(q, M) can be reformulated as

$$\min_{z \in \mathbb{R}^n} \quad z^\top(q + Mz) \tag{2.1.2a}$$
$$\text{s.t.} \quad z \geq 0, \tag{2.1.2b}$$
$$q + Mz \geq 0. \tag{2.1.2c}$$

Here the term $z^\top(q+Mz)$ is a penalty term for the complementarity constraints. As both z and $q + Mz$ are non-negative, the QP is bounded from below by 0 and the corresponding LCP has a solution if and only if the reformulation has a global optimal objective value of 0. We will denote this penalty function by P_C^Q, i.e.,

$$P_C^Q(z) := z^\top(q + Mz).$$

This solution approach is a good approach especially in the case of *monotone* LCPs, i.e., LCPs with a positive semi-definite matrix M, as in that case Reformulation 2.1.2 is convex.

2.1.2. Linear Complementarity Problems with Integer Constraints

In many practical cases, there might arise problems, for which some of the variables have to take on integer values, i.e., $z_i \in \mathbb{Z}$ for $i \in I \subseteq [n]$. These might be equilibrium quantities that can only be integer in practice or other combinatorial constraints. Such applications can be found in, e.g., Gabriel (2017), Gabriel et al. (2013a,b), and Weinhold and Gabriel (2020).

As mentioned before, optimization problems with integrality constraints are only decidable if the integer variables are bounded and we therefore concern ourselves primarily with binary constrained LCPs. We will still refer to the binary constraints also as integrality constraints.

The resulting mixed-integer linear complementarity problem (MILCP) is defined as follows.

Definition 2.1.3 (Mixed-Integer Linear Complementarity Problem).
Let $q \in \mathbb{R}^n$, $M \in \mathbb{R}^{n \times n}$, $I \subseteq [n]$ be given. The mixed-integer linear complementarity problem denoted by LCP(q, M, I) is the task to find a vector $z \in \mathbb{R}^n$ that satisfies

$$z \geq 0, \tag{2.1.3a}$$

$$q + Mz \geq 0, \tag{2.1.3b}$$

$$z^\top (q + Mz) = 0, \tag{2.1.3c}$$

$$z_I \in \{0, 1\}^I. \tag{2.1.3d}$$

or to show that no such vector exists.

The literature on MILCPs is rather sparse compared to the research done on continuous LCPs. Most publications are of theoretical nature. For example for purely integer LCPs, in Chandrasekaran et al. (1998) and Cunningham and Geelen (1998) the authors found a characterization for a matrix class that contains all matrices that ensure that for every integer vector q and every subset I, the corresponding MILCP always has a solution, if the corresponding continuous LCP has a solution. More recent results can be found in Dubey and Neogy (2018) and Sumita et al. (2018). There is also some literature of constructive nature that aims at actually computing a feasible point for a given MILCP. A first solution approach goes back to Pardalos and Nagurney (1990), with more recent results in Chandrasekaran et al. (1998) and very recent approaches in, e.g., Fomeni et al. (2019a,b), Gabriel (2017), and Gabriel et al. (2013a, 2021).
One takeaway from the theoretical publications is that the combination of complementarity and integrality constraints is very restrictive in the sense that for many problems there will be no solution that is both complementarity and integer feasible. Therefore, in many cases, the result of an MILCP is simply that no feasible point exists without giving any further insides into the problem. But there are cases for which not only actually feasible points are relevant, but also points that minimize some form of measure of infeasibility, if there is no feasible point. For our purposes, we want to relax both the integrality and complementarity constraints by including them in the objective function as penalty terms. We will not relax the linear constraints as they are not the source of infeasibility usually.

The existing techniques to solve MILCPs have different downsides. The first approaches that were found are mostly enumeration techniques and therefore do not necessarily perform well and only find points that are actually feasible, e.g., Pardalos and Nagurney (1990), while later techniques are mostly MILP reformulations that use big-M constraints for which appropriate big-Ms are not always known, e.g., Gabriel (2017) and Gabriel et al. (2013a), or continuous reformulations that only find local optima, e.g., Gabriel et al. (2021).

We strive to find a method that not only performs well and provably finds a solution if there is one, but that also finds a point that minimizes some measure of infeasibility for the integrality and complementarity constraints if there is no solution. In the case of *monotone* MILCPs we can use the the canonical extension of Reformulation 2.1.2, which is the following convex mixed-integer quadratic problem (MIQP):

Reformulation 2.1.4 (MIQP Penalty Reformulation of an MILCP)**.** The LCP(q, M, I) can be reformulated as

$$\min_{z \in \mathbb{R}^n} \quad z^\top (q + Mz) \tag{2.1.4a}$$

$$\text{s.t.} \quad z \geq 0, \tag{2.1.4b}$$

$$q + Mz \geq 0, \tag{2.1.4c}$$

$$z_I \in \{0, 1\}^I. \tag{2.1.4d}$$

Again, this reformulation is bounded from below by 0, as both z and $q + Mz$ are non-negative and the corresponding MILCP has a feasible point if and only if the MIQP reformulation has an optimal objective value of 0. This formulation not only finds solutions, but also points that are integer feasible and not necessarily complementarity feasible.

As we also want to find points that are neither integer nor complementarity feasible, we also want to include a penalty function for the integrality constraints. Therefore, we take a convex combination of penalty terms for both complementarity and integrality as the objective function. Such a reformulation was proposed in Gabriel et al. (2013a). As mentioned before, there the authors reformulate the MILCP as an MILP with additional binary variables and big-M constraints to model complementarity constraints. Note that we use the notation B for the large constants to avoid confusion with the LCP's matrix M. The respective MILP then reads as follows.

Reformulation 2.1.5. The LCP(q, M, I) can be reformulated with $\alpha \in (0, 1)$ and $B \in \mathbb{R}_{>0}$ large enough as

$$\min_{z, z', z'', \rho, \sigma} \quad \alpha \sum_{i=1}^{n} \rho_i + (1 - \alpha) \sum_{i \in I} \sigma_i \tag{2.1.5a}$$

$$\text{s.t.} \quad z \geq 0, \quad q + Mz \geq 0, \tag{2.1.5b}$$

$$z \leq Bz' + \rho, \tag{2.1.5c}$$

$$q + Mz \leq B(1 - z') + \rho, \tag{2.1.5d}$$

$$0 \leq z_I \leq z'' + \sigma, \tag{2.1.5e}$$

$$z'' - \sigma \leq z_I \leq 1, \tag{2.1.5f}$$

$$z \in \mathbb{R}^n, \quad z' \in \{0, 1\}^n, \quad z'' \in \{0, 1\}^I, \tag{2.1.5g}$$

$$\sigma \in \mathbb{R}^I_{\geq 0}, \quad \rho \in \mathbb{R}^n_{\geq 0}. \tag{2.1.5h}$$

Here, ρ_i is used to bound the violation of the complementarity constraint for each index, while σ_i bounds the violation of the binary constraints. The parameter α balances the emphasis between the penalization of either non-complementarity or non-integrality. 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.

We go a different route. As we focus on *monotone* MILCPs in this chapter, we can use the quadratic penalty function for complementarity P_C^Q from Reformulation 2.1.4 instead of using $\sum_{i=1}^n \rho_i$. As the penalty function for the integrality constraints, we use the function P_I from Section 1.1.2 instead of the term $\sum_{i\in I} \sigma_i$. The resulting reformulation readsas follows.

Reformulation 2.1.6 (Non-Convex Penalty Reformulation of a Monotone MILCP)**.** The LCP(q, M, I) can be reformulated with $\alpha \in (0,1)$ as

$$\min_{z\in\mathbb{R}^n} \quad \alpha z^\top (q+Mz) + (1-\alpha)\sum_{i\in I} \min\{z_i, 1-z_i\} \tag{2.1.6a}$$

$$\text{s.t.} \quad z \geq 0, \tag{2.1.6b}$$

$$q + Mz \geq 0, \tag{2.1.6c}$$

$$z_I \in [0,1]^I. \tag{2.1.6d}$$

As before, the reformulation is bounded from below by 0, as both z and $q+Mz$ are non-negative and the corresponding MILCP has a solution if and only if Reformulation 2.1.6 has an optimal objective value of 0. By including both penalty terms, the globally optimal point of the reformulation is either a feasible point of the corresponding MILCP or a point that minimizes that measure of infeasibility, if no solution exists. The above formulation is both non-convex and non-smooth and therefore difficult to solve to global optimality. In the following section, we will introduce a penalty branch-and-bound method to tackle the reformulation by exploiting the combinatorial structure of the otherwise problematic second penalty function.

2.2. The Penalty Branch-and-Bound Method

As mentioned before, in order to solve the problem stated in Reformulation 2.1.6, we want to employ a novel type of branch-and-bound method that exploits the combinatorial structure of the problematic non-convex terms in the objective function. In classic branch-and-bound methods, relaxations are solved and the feasible set is further divided by the addition of equality or inequality constraints. While the objective function stays the same in the entire branch-and-bound tree, the feasible set is further constrained as we go down the tree. In a sense, our method works the opposite way. For our method, the feasible set remains constant throughout the entire branching tree, but the objective function is altered at every node as additional penalty terms are added to incorporate the branching decisions. We will call this principle a penalty branch-and-bound (PBB). In the following, we will denote the feasible set of Reformulation 2.1.6 by Z, i.e.,

$$Z := \{z \in \mathbb{R}^n : z \geq 0,\ q + Mz \geq 0,\ z_I \in [0,1]^I\}.$$

Further, we denote the objective function of Reformulation 2.1.6 by f, i.e.,

$$f(z) := \alpha z^\top (q + Mz) + (1-\alpha) \sum_{i \in I} \min\{z_i, 1 - z_i\}.$$

2.2.1. Branching

In classic branch-and-bound methods, the relaxations solved during the process are obtained by relaxing the non-convex integer constraints. In our method, we obtain the relaxations to solve by relaxing the non-convex integer penalty term of the objective function. Hence, the first problem to solve is the following.

Definition 2.2.1 (Root Node Problem of PBB for Monotone MILCPs)**.** The root node problem of the PBB for the LCP(q, M, I) is defined as

$$\begin{aligned} \min_{z \in \mathbb{R}^n} \quad & \alpha z^\top (q + Mz) && \text{(2.2.1a)} \\ \text{s.t.} \quad & z \in Z. && \text{(2.2.1b)} \end{aligned}$$

Note, that because M is positive semi-definite this problem is a convex QP and therefore tractable.

Then, very similar to classic branch-and-bound methods, two child nodes are created, both with a different objective function. For the left child node, the term $(1-\alpha)z_j$ is added, where $j \in I$ is an index, for which the optimal solution of Problem 2.2.1 has a fractional value. For the right child, the term $(1-\alpha)(1-z_j)$ is added and the two resulting child node problems are the following.

Definition 2.2.2 (Left Child Problem of the Root Node of PBB for Monotone MILCPs)**.** The left child problem of the PBB for the LCP(q, M, I) is defined as

$$\begin{aligned} \min_{z \in \mathbb{R}^n} \quad & \alpha z^\top (q + Mz) + (1-\alpha) z_j && \text{(2.2.2a)} \\ \text{s.t.} \quad & z \in Z. && \text{(2.2.2b)} \end{aligned}$$

Definition 2.2.3 (Right Child Problem of the Root Node of PBB for Monotone MILCPs)**.** The right child problem of the PBB for the LCP(q, M, I) is defined as

$$\begin{aligned} \min_{z \in \mathbb{R}^n} \quad & \alpha z^\top (q + Mz) + (1-\alpha)(1 - z_j) && \text{(2.2.3a)} \\ \text{s.t.} \quad & z \in Z. && \text{(2.2.3b)} \end{aligned}$$

The idea is that, by taking the minimum of these problem, we can model the non-convex function $\min\{z_j, 1 - z_j\}$, i.e.,

$$\begin{aligned} &\min_{z \in Z} \{\alpha z^\top (q + Mz) + (1 - \alpha) \min\{z_j, 1 - z_j\}\} \\ =& \min \left\{ \min_{z \in Z} \alpha z^\top (q + Mz) + (1 - \alpha) z_j, \min_{z \in Z} \alpha z^\top (q + Mz) + (1 - \alpha)(1 - z_j) \right\}. \end{aligned} \tag{2.2.4}$$

These problems are then solved independently and the addition of left and right child nodes is continued. Therefore, at an arbitrary node of the tree a certain combination of these penalty terms for different indices has been added. In the following, we denote $I_0 \subseteq I$ to be the set of indices $j \in I$ for which $(1 - \alpha)z_j$ has been added and $I_1 \subseteq I$ to be the set of indices $j \in I$ for which $(1 - \alpha)(1 - z_j)$ has been added. We can then uniquely identify every node of the tree by those sets and we denote a node as $N = (I_0, I_1)$. The corresponding node problem reads as follows.

Definition 2.2.4 (Node Problem of PBB for Monotone MILCPs)**.** The node problem of PBB for monotone MILCP at node $N = (I_0, I_1)$ is defined as

$$\min_{z \in \mathbb{R}^n} \quad \alpha z^\top (q + Mz) + (1 - \alpha) \sum_{i \in I_0} z_i + (1 - \alpha) \sum_{i \in I_1} (1 - z_i) \tag{2.2.5a}$$

$$\text{s.t.} \quad z \in Z. \tag{2.2.5b}$$

Note that as we only add linear terms, every node problem in the tree is a convex QP. In the following, we will call the addition of a left child node "downwards branching" and the addtion of a right child "upwards branching" and we will denote the objective function of the node problem of $N = (I_0, I_1)$ by f_N, i.e.,

$$f_N(z) := \alpha z^\top (q + Mz) + (1 - \alpha) \sum_{i \in I_0} z_i + (1 - \alpha) \sum_{i \in I_1} (1 - z_i).$$

Now, we show that enumerating all possible partitions (I_0, I_1) of I, i.e., $I = I_0 \cup I_1$ with $I_0 \cap I_1 = \emptyset$, yields the global optimum of Problem 2.1.6. 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 optimal solution of Reformulation 2.1.6.

Lemma 2.2.5. *Let z^* be an optimal solution of Problem 2.1.6. Then, it holds*

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

Proof. Note that the feasible set does not depend on N. Hence, all optimal points are feasible for all nodes. Let $N^* = (I_0^*, I_1^*)$ be the leaf with $I_0^* := \{i \in I \colon z_i^* \le 1 - z_i^*\}$ and

$I_1^* := \{i \in I : z_i^* > 1 - z_i^*\}$. We then have

$$\begin{aligned} f(z^*) &= \alpha(z^*)^\top (q + Mz^*) + (1-\alpha) \sum_{j \in I} \min\left\{z_j^*, 1 - z_j^*\right\} \\ &= \alpha(z^*)^\top (q + Mz^*) + (1-\alpha) \sum_{j \in I_0^*} z_j^* + (1-\alpha) \sum_{j \in I_1^*} (1 - z_j^*) \\ &= f_{N^*}(z^*) \geq f_{N^*}(z_{N^*}^*). \end{aligned}$$

Hence,

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

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,

$$\begin{aligned} &\alpha(z_{N'}^*)^\top (q + Mz_{N'}^*) + (1-\alpha) \sum_{j \in I_0'} (z_{N'}^*)_j + (1-\alpha) \sum_{j \in I_1'} (1 - (z_{N'}^*)_j) \\ < \; &\alpha(z_{N'}^*)^\top (q + Mz_{N'}^*) + (1-\alpha) \sum_{j \in I} \min\left\{(z_{N'}^*)_j, 1 - (z_{N'}^*)_j\right\}. \end{aligned}$$

This implies

$$\sum_{j \in I_0'} \left((z_{N'}^*)_j - \min\left\{(z_{N'}^*)_j, 1 - (z_{N'}^*)_j\right\}\right) + \sum_{j \in I_1'} \left(1 - (z_{N'}^*)_j - \min\left\{(z_{N'}^*)_j, 1 - (z_{N'}^*)_j\right\}\right) < 0,$$

which is impossible as

$$(z_{N'}^*)_j \geq \min\left\{(z_{N'}^*)_j, 1 - (z_{N'}^*)_j\right\}$$

and

$$1 - (z_{N'}^*)_j \geq \min\left\{(z_{N'}^*)_j, 1 - (z_{N'}^*)_j\right\}.$$

Hence,

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

holds and the claim follows. □

This is the first necessary step in order to prove the overall correctness of our method.

2.2.2. Bounding

We now want to show ways to avoid fully enumerating all exponentially many combinations. 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 2.1.6. 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_{inc}^* the incumbent, i.e., the point so that $f(z_{\text{inc}}^*)$ constitutes the best upper bound for Problem 2.1.6 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.

Lemma 2.2.6. *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,

$$\begin{aligned}
f_{N'}(z_{N'}^*) &= \alpha\ (z_{N'}^*)^\top (q + Mz_{N'}^*) + (1-\alpha) \sum_{j \in I_0'} (z_{N'}^*)_j + (1-\alpha) \sum_{j \in I_1'} (1 - (z_{N'}^*)_j) \\
&= \alpha\ (z_{N'}^*)^\top (q + Mz_{N'}^*) + (1-\alpha) \sum_{j \in I_0} (z_{N'}^*)_j + (1-\alpha) \sum_{j \in I_1} (1 - (z_{N'}^*)_j) \\
&\quad + (1-\alpha) \sum_{j \in I_0' \setminus I_0} (z_{N'}^*)_j + (1-\alpha) \sum_{j \in I_1' \setminus I_1} (1 - (z_{N'}^*)_j) \\
&\geq \alpha\ (z_{N'}^*)^\top (q + Mz_{N'}^*) + (1-\alpha) \sum_{j \in I_0} (z_{N'}^*)_j + (1-\alpha) \sum_{j \in I_1} (1 - (z_{N'}^*)_j) \\
&= f_N(z_{N'}^*) \geq f_N(z_N^*).
\end{aligned}$$

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. $\square$

This allows us to stop adding child nodes in certain cases. If we compute an optimal solution z_N^* at a node N that has the property

$$f_N(z_N^*) \geq f(z_{\text{inc}}^*), \tag{2.2.6}$$

we know that any node in the subtree rooted in N including the leaves cannot yield a better solution than the best solution already known. This is the penalty-equivalent of suboptimality pruning known from classic branch-and-bound methods. Obviously, we can also stop adding nodes in the case, that there are no fractional variables left that are supposed to be integer, which is the equivalent to feasibility pruning in classic branch-and-bound methods. Note, that as the feasible set does not change throughout the process, there is no equivalent to infeasibility pruning in classic branch-and-bound methods for now.

2.2.3. Algorithmic Description

We are now ready to give the full algorithmic description of our penalty branch-and-bound algorithm.

Algorithm 2 A Penalty Branch-and-Bound Algorithm for MILCPs

Input: $q \in \mathbb{R}^n$, $M \in \mathbb{R}^{n\times n}$, $I \subseteq [n]$, $\alpha \in (0,1)$
Output: A global optimum z^* of Problem 2.1.6.
Set $\mathcal{N} \leftarrow \{(\emptyset,\emptyset)\}$.
Set $f_{\text{inc}} \leftarrow \infty$, $z^*_{\text{inc}} \leftarrow$ none.
while $\mathcal{N} \neq \emptyset$ **do**
  Choose $N = (I_0, I_1) \in \mathcal{N}$ and set $\mathcal{N} \leftarrow \mathcal{N} \setminus \{N\}$.
  Compute $z^*_N \in \operatorname{argmin}\{f_N(z) : z \in Z\}$.
  if $f(z^*_N) < f_{\text{inc}}$ **then**
    Set $z^*_{\text{inc}} \leftarrow z^*_N$ and set $f_{\text{inc}} \leftarrow f(z^*_N)$.
  if $f_N(z^*_N) < f_{\text{inc}}$ **and if** there is a $j \in I \setminus (I_0 \cup I_1)$ with $(z^*_N)_j \notin \{0,1\}$ **then**
    Choose $j \in I \setminus (I_0 \cup I_1)$ with $(z^*_N)_j \notin \{0,1\}$.
    Set $\mathcal{N} \leftarrow \mathcal{N} \cup \{(I_0 \cup \{j\}, I_1), (I_0, I_1 \cup \{j\})\}$.
return z^*_{inc}

Theorem 2.2.7. *Algorithm 2 terminates after finitely many steps with a global optimal solution of Problem 2.1.6.*

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 $f_N(z^*_N) < f(z^*_{\text{inc}})$ always holds in the second if-clause. Then, the correctness of the algorithm follows from Lemma 2.2.5 as we iterate through the complete branch-and-bound tree. Finally, in the cases in which $f_N(z^*_N) \geq f(z^*_{\text{inc}})$ holds, the nodes that are not added can be excluded due to Lemma 2.2.6. □

2.3. Further Enhancements of the Method

As mentioned in Section 1.1.1, there are some ways to enhance the performance of classic branch-and-bound methods. In this section we investigate possible approaches to achieve analogous improvements for our method. In Section 2.3.1 we investigate different ways to choose the next branching index $j \in I$ and, in Section 2.3.2, ways to choose the next subproblem $N \in \mathcal{N}$ to solve, both of which have not been specified in Algorithm 2, as they are not relevant for the correctness of the method. In Section 2.3.3 we briefly discuss the possibilities of warmstarting the different node problems and in Section 2.3.4 we propose two different types of cutting planes.

2.3.1. Choosing the Branching Index

In every node $N = (I_0, I_1)$ of our penalty branch-and-bound method, we need to choose an index $j \in I \setminus (I_0 \cup I_1)$ for which $(z^*_N)_j$ is fractional to define the objective functions of the problems in the child nodes. There are various ways to do that. 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 $1/2$.

Pseudocost Branching

Pseudocost branching is a technique commonly used in branch-and-bound algorithms for mixed-integer programs that goes back to Benichou et al. (1971).

The idea 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:

$$\varphi^1_{N,j} := \frac{f(z^*_{N_1}) - f(z^*_N)}{\lceil (z^*_{N_1})_j \rceil - (z^*_{N_1})_j}.$$

Here, N_1 is the child of N created by upwards branching. We denote by ψ^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 ψ^1_j as

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

Analogously, we define $\varphi^0_{N,j}$ and ψ^0_j for branching downwards on variable $j \in I$, with the only difference being the denominator of $\varphi^0_{N,j}$. The average gain is then calculated as

$$\begin{aligned} s_j := {} & \mu \min\left\{\psi^0_j \cdot ((z^*_{N_0})_j - \lfloor (z^*_{N_0})_j \rfloor),\, \psi^1_j \cdot (\lceil (z^*_{N_1})_j \rceil - (z^*_{N_1})_j)\right\} \\ & + (1-\mu) \max\left\{\psi^0_j \cdot ((z^*_{N_0})_j - \lfloor (z^*_{N_0})_j \rfloor),\, \psi^1_j \cdot (\lceil (z^*_{N_1})_j \rceil - (z^*_{N_1})_j)\right\} \end{aligned}$$

with $\mu \in (0, 1)$. The pseudocost-based branching candidate is the index $j \in I$ with the largest score s_j. At the beginning of our branch-and-bound, we initialize the average $\psi^{0,1}_j$

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 $i \neq j$.

MIQP-Based Branching

We have also devised a second technique. In a pre-processing phase of the algorithm we can sort the indices $j \in I$ in a way, so that we branch on the indices for which we expect good lower bounds first. For every index $j \in I$, we solve the following MIQP with a single integer variable:

$$\begin{aligned} \min_{z \in \mathbb{R}^n} \quad & z^\top (q + Mz) && \text{(2.3.1a)} \\ \text{s.t.} \quad & q + Mz \geq 0, \ z \geq 0, && \text{(2.3.1b)} \\ & z_j \in \{0, 1\}. && \text{(2.3.1c)} \end{aligned}$$

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 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 (2.3.1). Moreover, infeasible problems are formally assigned the objective function value ∞. The resulting branching strategy then chooses the branching candidate at the top of the list while skipping all integer-feasible indices as well as all indices that have been branched on already. Additionally, we can use the optimal solutions of each of these MIQPs to constitute a first upper bound on our branch-and-bound process, as each of the points is also feasible for our method.

2.3.2. Choosing the Next Subproblem to Solve

For the selection of the next node $N \in \mathcal{N}$ to solve we propose one technique, which we will call the “lower bound push” strategy. In our implementation we test that strategy against both breadth- and depth-first search as benchmarks.

Lower Bound Push

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 a 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 take the left child if $(z_N)_j^* \leq 0.5$ with $j \in I$ being the index that has been branched on last and the right child otherwise. 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.

2.3.3. Warmstarting the Node Problems

Recall that all nodes of the search tree share the same feasible set and that the objective functions change only slightly from a parent node to its child nodes. This allows for warmstarting 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.

2.3.4. First Types of Valid Inequalities

In this section, we propose two classes of inequalities. In classic branch-and-bound methods, valid inequalities are used to cut off points that are feasible for the relaxations of the problem but not the problem itself in order to tighten the relaxation. In our method, the feasible set does not change, which is why we need different criteria. The cuts we propose are not valid for the overall Problem 2.1.6 in the classic sense but are locally valid. The first class of valid inequalities are called “simple cuts”. They are used to split the feasible set according to the branching decisions that were already made. The second class are called “optimality cuts”. They are supposed to cut of points, that do not fulfill necessary optimality conditions.

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 $z_j \geq 0.5$. We first show that the minimum among the optimal solutions of the leaves when including the simple cuts is the optimal solution of Problem 2.1.6.

Lemma 2.3.1. *Let*

$$z_N^* \in \operatorname{argmin}\{f_N(z) \colon z \in Z,\, z_{I_0} \leq 0.5,\, z_{I_1} \geq 0.5\}$$

be an optimal solution at node N when all simple cuts are included. Then,

$$f(z^*) = \min\{f_N(z_N^*) \colon N = (I_0, I_1) \text{ with } I_0 \cup I_1 = I\}$$

holds.

Proof. Let $N^* = (I_0^*, I_1^*)$ be the leaf defined by $I_0^* := \{j \in I \colon z_j^* \leq 1 - z_j^*\}$ and $I_1^* := \{j \in I \colon z_j^* > 1 - z_j^*\}$. We then have

$$\begin{aligned} f(z^*) &= \alpha(z^*)^\top(q + Mz^*) + (1-\alpha)\sum_{i \in \mathcal{I}} \min\{z_i^*, 1 - z_i^*\} \\ &= \alpha(z^*)^\top(q + Mz^*) + (1-\alpha)\sum_{j \in I_0^*} z_j^* + (1-\alpha)\sum_{j \in I_1^*}(1 - z_j^*) \\ &= f_{N^*}(z^*) \geq f_{N^*}(z_{N^*}^*). \end{aligned}$$

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,

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

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,

$$\begin{aligned} &\alpha(z_{N'}^*)^\top (q + M z_{N'}^*) + (1-\alpha) \sum_{j \in I_0'} (z_{N'}^*)_j + (1-\alpha) \sum_{j \in I_1'} (z_{N'}^*)_j \\ &< \alpha(z_{N'}^*)^\top (q + M z_{N'}^*) + (1-\alpha) \sum_{j \in \mathcal{I}} \min \{(z_{N'}^*)_j, 1 - (z_{N'}^*)_j\} . \end{aligned}$$

This implies

$$\sum_{j \in I_0'} \left(z_{N',j}^* - \min \{(z_{N'}^*)_j, 1-(z_{N'}^*)_j\} \right) + \sum_{j \in I_1'} (1 - (z_{N'}^*)_j - \min \{(z_{N'}^*)_j, 1-(z_{N'}^*)_j\}) < 0,$$

which is a contradiction by definition. Hence,

$$f(z^*) \leq \min \{f_N(z_N^*) \colon N = (I_0, I_1) \text{ with } I_0 \,\dot\cup\, I_1 = I\}$$

holds and the claim follows. □

As a second result, we show that Lemma 2.2.6 is also valid when simple cuts are used in the branch-and-bound method.

Lemma 2.3.2. *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. Note that $(z_{N'}^*)_j \geq 0$ and $1 - (z_{N'}^*)_j \geq 0$ are valid on the feasible set and 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. By definition, we then have

$$\begin{aligned}
f_{N'}(z^*_{N'}) &= \alpha(z^*_{N'})^\top(q + Mz^*_{N'}) + (1-\alpha)\sum_{j\in I'_0}(z^*_{N'})_j + (1-\alpha)\sum_{j\in I'_1}(1-(z^*_{N'})_j)\\
&= \alpha(z^*_{N'})^\top(q + Mz^*_{N'}) + (1-\alpha)\sum_{j\in I_0}(z^*_{N'})_j + (1-\alpha)\sum_{j\in I_1}(1-(z^*_{N'})_j)\\
&\quad + (1-\alpha)\sum_{j\in I'_0\setminus I_0}(z^*_{N'})_j + (1-\alpha)\sum_{j\in I'_1\setminus I_1}(1-(z^*_{N'})_j)\\
&\geq \alpha(z^*_{N'})^\top(q + Mz^*_{N'}) + (1-\alpha)\sum_{j\in I_0}(z^*_{N'})_j + (1-\alpha)\sum_{j\in I_1}(1-(z^*_{N'})_j)\\
&= f_N(z^*_{N'}) \geq f_N(z^*_N).
\end{aligned}$$

Hence, every feasible point of N' is also feasible for N. □

Theorem 2.3.3. *Algorithm 2 remains correct when simple cuts*

$$z_j \leq 0.5 \textit{ for all } j \in I_0, \quad z_j \geq 0.5 \textit{ for all } j \in I_1$$

are added at any node $N = (I_0, I_1)$.

Proof. From Lemma 2.3.1, we know that the optimal solution of Problem 2.1.6 is the optimal solution of a leaf node. From Lemma 2.3.2, 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^*_{inc} of Problem 2.1.6 and some node N for which

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

holds, we know that z^*_{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 2 remains correct when simple cuts are used. □

Optimality Cuts

In order to define the optimality cuts, we use the necessary optimality conditions for Problem 2.1.6; see, e.g., Corollary 3.68 in Beck (2017). Let $z^* \in Z$ be an optimal solution of Problem 2.1.6, then $g \in \partial f(z^*)$ exists such that

$$g^\top(z - z^*) \geq 0 \quad \text{for all } z \in Z.$$

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 $\bar{g}^\top z' \geq g^\top z'$ and $\tilde{g}^\top z \leq g^\top z$ holds, the following inequality is also valid:

$$\bar{g}^\top z' \geq \tilde{g}^\top z.$$

This will be necessary to convexify the valid inequality.

Lemma 2.3.4. *Let $z' \in Z$ be a feasible solution and let $N = (I_0, I_1)$. Then,*

$$\begin{aligned} &\alpha z^\top (q + 2Mz) + (1-\alpha) \sum_{j \in I_0} z_j + (1-\alpha) \sum_{j \in I_1} (1 - z_j) - (1-\alpha)|I \setminus I_0| \\ &\leq \alpha (z')^\top (q + 2Mz) + (1-\alpha) \sum_{j \in I \setminus I_1} z'_j \end{aligned}$$

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

$$g_i = \alpha q_i + \alpha \sum_{j \in [n]} 2M_{i,j} z_j \begin{cases} +(1-\alpha), & \text{for } z_i < 0.5,\ i \in I, \\ -(1-\alpha), & \text{for } z_i > 0.5,\ i \in I, \\ +(1-\alpha) y_i, & \text{for } z_i = 0.5,\ i \in I, \\ +0, & \text{for } i \notin I, \end{cases}$$

for some $y_i \in [-1, 1]$.

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

$$\begin{aligned} g^\top z &= \alpha z^\top (q + 2Mz) + (1-\alpha) \left(\sum_{\substack{i \in I: \\ z_i < 0.5}} z_i - \sum_{\substack{i \in I: \\ z_i > 0.5}} z_i + \sum_{\substack{i \in I: \\ z_i = 0.5}} y_i^g z_i \right) \\ &\geq \alpha z^\top (q + 2Mz) + (1-\alpha) \left(\sum_{\substack{i \in I: \\ z_i < 0.5}} z_i - \sum_{\substack{i \in I: \\ z_i > 0.5}} z_i - \sum_{\substack{i \in I: \\ z_i = 0.5}} z_i \right) \\ &= \alpha z^\top (q + 2Mz) + (1-\alpha) \left(\sum_{\substack{i \in I: \\ z_i < 0.5}} z_i + \sum_{\substack{i \in I: \\ z_i \geq 0.5}} (1 - z_i) - \sum_{\substack{i \in I: \\ z_i \geq 0.5}} 1 \right) \\ &\geq \alpha z^\top (q + 2Mz) + (1-\alpha) \sum_{i \in I} \min\{z_i, 1 - z_i\} - (1-\alpha)|I| \\ &\geq \alpha z^\top (q + 2Mz) + (1-\alpha) \sum_{i \in I_0} z_i + (1-\alpha) \sum_{i \in I_1} (1 - z_i) - (1-\alpha)|I|. \end{aligned}$$

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:

$$\begin{aligned}
g^\top z' &= \alpha(z')^\top(q + 2Mz) + (1-\alpha)\left(\sum_{\substack{i\in I:\\ z_i<0.5}} z_i' - \sum_{\substack{i\in I:\\ z_i>0.5}} z_i' + \sum_{\substack{i\in I:\\ z_i=0.5}} y_i^g z_i'\right)\\
&\le \alpha(z')^\top(q + 2Mz) + (1-\alpha)\left(\sum_{\substack{i\in I:\\ z_i<0.5}} z_i' - \sum_{\substack{i\in I:\\ z_i>0.5}} z_i' + \sum_{\substack{i\in I:\\ z_i=0.5}} z_i'\right)\\
&= \alpha(z')^\top(q + 2Mz) + (1-\alpha)\left(\sum_{\substack{i\in I:\\ z_i\le 0.5}} z_i' - \sum_{\substack{i\in I:\\ z_i>0.5}} z_i'\right)\\
&\le \alpha(z')^\top(q + 2Mz) + (1-\alpha)\sum_{i\in I\setminus I_1} z_i' - (1-\alpha)\sum_{\substack{i\in I_1:\\ z_i\neq 0.5}} z_i'\\
&\le \alpha(z')^\top(q + 2Mz) + (1-\alpha)\sum_{i\in I\setminus I_1} z'.
\end{aligned}$$

The combination of the two inequalities yields the lemma. □

2.4. Numerical Results

In Section 2.3 we proposed various ways to improve the overall performance of our method. In this section, we will present the results of numerical experiments we conducted to compare the different techniques performance-wise. We tested all types of enhancements independently of each other. For every test we take the best settings from previous tests together with "standard" settings for untested parameters. We start with a test of the different branching rules in Section 2.4.1, followed by a test of the node selection strategies in Section 2.4.2, different warm starting techniques in Section 2.4.3, and different strategies for the inclusion of valid inequalities in Section 2.4.4. Afterwards, we test our method with the best setting we identified against two benchmark approaches.

We used Python 3.7 to implement the penalty branch-and-bound method presented in Section 2.2. All node problems are solved with the QP solver of Gurobi 9.1.2 and all the tests were run on an Intel Xeon CPU E5-2699 v4 @ 2.20 GHz (88 cores) with 756 GB RAM. In this section, we refer to the implementation of Algorithm 2 as MILCP-PBB. For our tests, we consider instances that we randomly generated as follows. The positive semi-definite matrices $M \in \mathbb{R}^{n\times n}$ have been created using the sprandsym function of MATLAB for sizes

$$n \in \{50, 100, 150, 200, 250, 300, 350, 400, 450, 500\}.$$

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 2.1.3. Then, it satisfies

(i) Feasibility w.r.t. Z: $z^* \in Z$,

(ii) Integrality: $z_i^* \in \{0, 1\}$ for all $i \in I$,

(iii) 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. 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) as well as tables with the most important statistical measures. For the tables, we aggregated all instances that have been solved by all parameter settings or solution approaches for the specific test w.r.t. the instance size. The first column always states the dimension n of the problem. The second column contains the arithmetic mean of node counts and running times respectively for all instances solved by every parameterization. The next columns contain the median, the minimum, and the maximum value of the data set. The sixth and seventh column contain the 0.25-quantile, i.e., the node count or running time after which 25 % of instances were solved, as well as the 0.75-quantile. The next column contains the geometric shifted mean. The shift is 100 for the node counts and 10 for the running times. The last column contains the percentage of instances solved to global optimality for the parameterization and instance size. The best value for every measure and instance size among all tables for that test is printed in bold font. The table of the winning setting, i.e., the best performing parameterization, is included in this section whereas the tables of the other settings are included in Appendix B. The timelimit for these tests is set to 1 h.

2.4.1. The Impact of Different Branching Rules

We now compare the performance of MILCP-PBB when equipped with the four different branching rules described in Section 2.3.1. For these tests, the node selection strategy is set

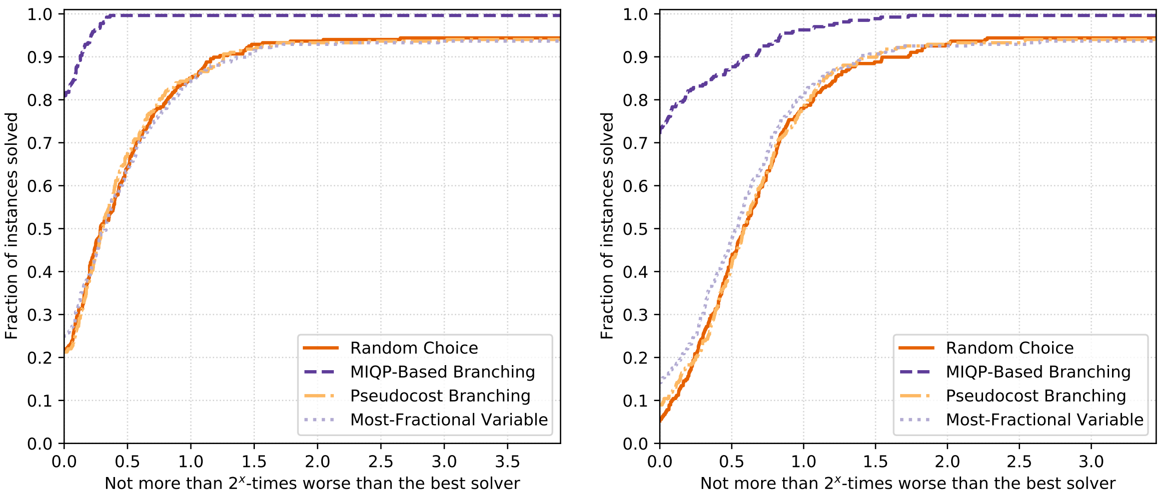


Figure 2.1.: Performance profiles for the number of branch-and-bound nodes (left) and the running time (right) of all branching rules

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 32 instances from the test set since no parameterization is able to solve them within the time limit. Figure 2.1 displays the performance profiles w.r.t. the required number of branch-and-bound nodes (left figure) and running times (right figure). 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 in terms of the required number of nodes, the running time, and also in terms of the overall number of solved instances. 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 a little less 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.

Similar conclusions can be drawn from the statistical measures as displayed in the Table 2.1 (and Tables B.1–B.3 in the appendix). In comparison of all tables one sees that, except for the minimum running time, the MIQP-based branching rule outperforms the other approaches w.r.t. almost every other measure and every instance size.

2.4.2. The Impact of Different Node Selection Strategies

We now compare the three node selection strategies described in Section 2.3.2. To this end, we use the MIQP-based branching strategy, while warmstarts and valid inequalities

Table 2.1.: Aggregated node counts (top) and runtimes (bottom) for the branching rule test with MIQP-based branching

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
50	13.9	9.0	**3.0**	**37.0**	**5.0**	**21.0**	13.4	**100**
100	**17.5**	**13.0**	**3.0**	73.0	**7.0**	**21.0**	**16.7**	**100**
150	**38.5**	**33.0**	**7.0**	**139.0**	**14.0**	**44.5**	**35.4**	**100**
200	**60.1**	**44.0**	**15.0**	235.0	**31.0**	**82.5**	**55.3**	**100**
250	**116.5**	**96.0**	**5.0**	**353.0**	**68.5**	**132.5**	**100.7**	**100**
300	**273.9**	**155.0**	**15.0**	1199.0	**95.5**	**292.0**	**203.3**	**100**
350	**549.8**	**331.0**	**7.0**	**2705.0**	**80.0**	**729.0**	**333.0**	**100**
400	**426.7**	**248.0**	**47.0**	**1245.0**	**76.5**	**750.5**	**289.7**	**80**
450	**408.3**	**349.0**	**51.0**	**1713.0**	**139.0**	**511.0**	**305.2**	**67**
500	**544.1**	**543.0**	**71.0**	**1043.0**	**342.0**	**734.0**	**444.9**	**40**
50	0.3	0.3	**0.1**	**0.5**	0.2	0.3	0.3	**100**
100	**2.4**	1.7	0.6	**9.4**	1.1	**2.9**	**2.2**	**100**
150	**15.1**	**12.6**	3.3	**54.0**	**5.2**	**18.2**	**12.7**	**100**
200	**45.5**	**31.5**	13.2	**147.6**	**24.0**	**58.1**	**38.8**	**100**
250	**149.8**	**112.8**	16.3	**504.9**	**75.4**	**180.4**	**110.1**	**100**
300	**448.8**	**291.2**	48.5	**1918.0**	**162.0**	**550.5**	**306.8**	**100**
350	**889.3**	**574.3**	54.2	**2853.7**	**211.5**	**1575.2**	**559.2**	**100**
400	**821.3**	**726.8**	**191.3**	**2429.7**	**302.9**	**1260.8**	**620.7**	**80**
450	**909.3**	**941.1**	260.4	**2325.1**	**469.1**	**1176.4**	**766.6**	**67**
500	**1106.9**	**1197.4**	355.0	**1666.5**	**870.6**	**1394.2**	**1000.5**	**40**

are disabled. We exclude 54 instances from the set since no parameterization of our method is able to solve them within the time limit. Based on Figure 2.2, 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. Especially regarding the required number of branch-and-bound nodes, no parameterization seems to have an advantage. Regarding the running time, the lower-bound-push strategy seems to be slightly worse, while breadth-first and depth-first search are very close in comparison with a slight advantage for the depth-first search. Again, this is due to the higher computational cost for the ordering of the nodes. As the depth-first search also solves slightly more instances, we choose it for our "best-setting" implementation of MILCP-PBB.

The statistical measures we present in Tables 2.2, B.4, and B.5 support these conclusions. For most measures the depth-first search strategy performs best, followed by the breadth-first search strategy. But nevertheless, for all measures and instance sizes, the differences are rather small.

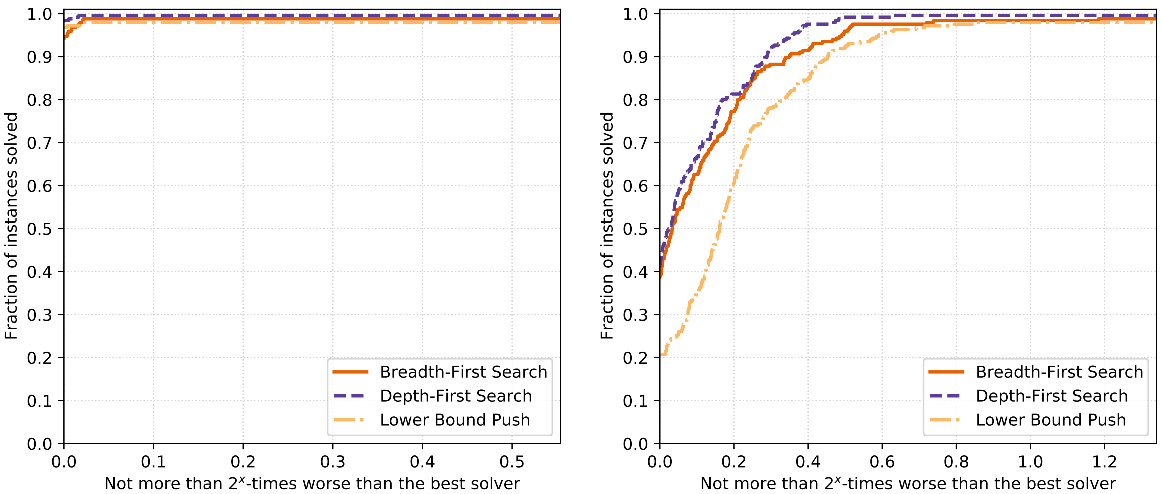


Figure 2.2.: Performance profiles on the number of branch-and-bound nodes (left) and the running time (right) of all node selection strategies.

Table 2.2.: Aggregated node counts (top) and runtimes (bottom) for the node selection test with depth-first search

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
50	**13.9**	**8.0**	**3.0**	**37.0**	**5.0**	**21.0**	**13.3**	**100**
100	**16.1**	**10.0**	**3.0**	**73.0**	**7.0**	**21.0**	**15.3**	**100**
150	**35.6**	**24.0**	**7.0**	**139.0**	**13.5**	**42.5**	**32.6**	**100**
200	**59.5**	**43.0**	**15.0**	**235.0**	**31.0**	**82.5**	**54.9**	**100**
250	**112.3**	**96.0**	**5.0**	**353.0**	**67.5**	**131.0**	**97.6**	**100**
300	**193.3**	**141.0**	**15.0**	**741.0**	**95.0**	**254.5**	**161.7**	**100**
350	**277.0**	**144.0**	**7.0**	**735.0**	**79.0**	**479.0**	**203.1**	**77**
400	**351.2**	**271.0**	**47.0**	**813.0**	**78.0**	**613.0**	**259.3**	**70**
450	314.7	345.0	**51.0**	**647.0**	**126.5**	**473.5**	**263.7**	**47**
500	461.3	519.0	**71.0**	**923.0**	**265.50**	546.0	381.8	**23**
50	**0.3**	**0.2**	**0.1**	**0.4**	**0.2**	**0.3**	**0.3**	**100**
100	**4.5**	**2.6**	1.4	24.1	2.0	5.4	**4.0**	**100**
150	27.1	**18.1**	6.4	97.0	10.6	32.6	22.1	**100**
200	91.9	**65.1**	30.4	299.9	**49.8**	**111.6**	**78.5**	**100**
250	249.5	221.9	37.5	**788.4**	156.2	**286.80**	198.9	**100**
300	**566.8**	**427.2**	132.4	**1889.0**	**266.2**	769.5	**454.1**	**100**
350	**1064.1**	**543.2**	**102.3**	**2983.5**	401.0	**1623.7**	**716.8**	**77**
400	1457.7	**1146.4**	309.1	**3215.6**	**419.6**	2474.4	**1060.7**	**70**
450	**1540.1**	**1468.0**	**369.0**	**2823.5**	**654.3**	2489.2	**1243.7**	**47**
500	2098.4	2272.3	553.1	**3178.9**	1553.9	2817.7	1821.9	**23**

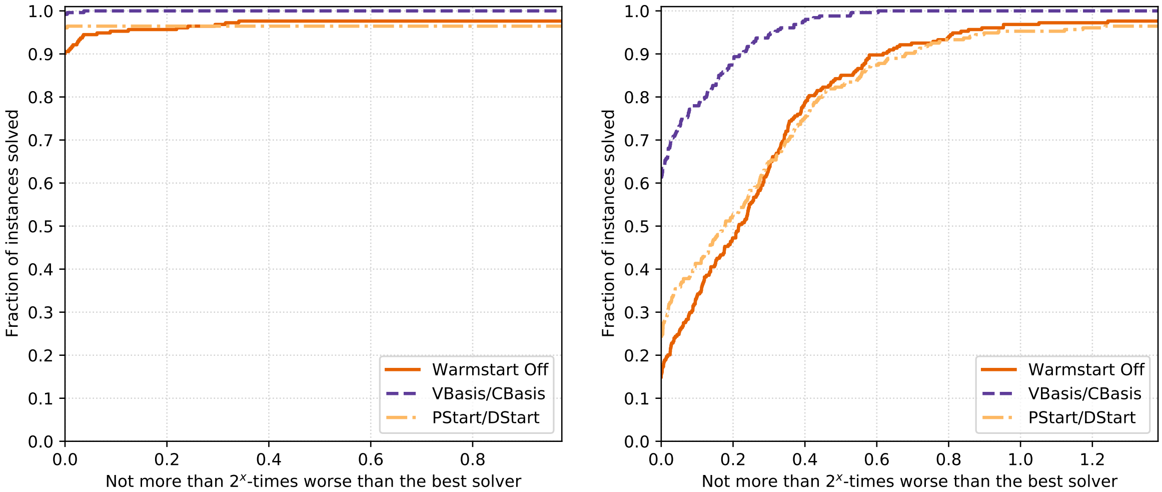


Figure 2.3.: Performance profiles for the number of branch-and-bound nodes (left) and the running time (right) of the warmstart test.

2.4.3. The Impact of Warmstarts

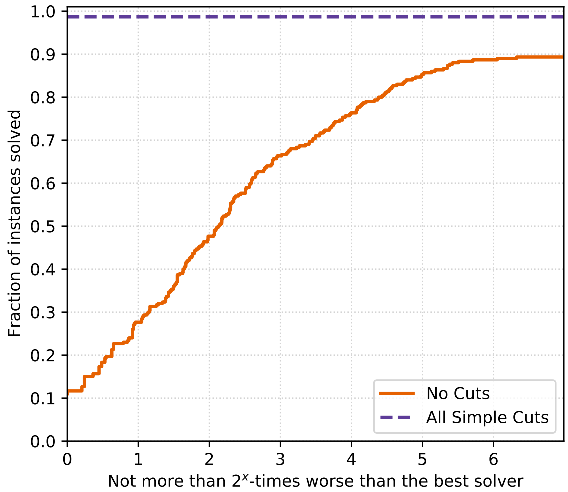
We now compare the performance of MILCP-PBB with and without warmstarts. To this end, we use the MIQP-based approach branching rule, the depth-first search node selection strategy, and avoid the use of any valid inequalities. We tried two different techniques within Gurobi to warm start the node problems. First, we used the Gurobi attributes VBasis and CBasis, i.e., we started every node problem with the optimal basis of its parent node. Second, we used the attributes PStart and DStart, where the optimal basis vector of the parent node is computed from the optimal solution. 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 that we detected 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 46 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 2.3 (right). Especially the use of parameters VBasis and CBasis have a big impact, which is expected as it is not needed to compute the basis vector first. 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 2.3 (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, effects the overall search tree. The same can be seen in Tables 2.3, B.6, and B.7. For the node counts, differences are not remarkably large with a slight advantage for the warmstarted methods on most instances. With respect to running times, the warmstarting strategy using VBasis and CBasis gives a significant advantage for all measures and instance sizes.

Table 2.3.: Aggregated node counts (top) and runtimes (bottom) for the warmstart test using VBasis/CBasis

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
50	**13.9**	**8.0**	**3.0**	**37.0**	**5.0**	**21.0**	**13.3**	**100**
100	**16.1**	**10.0**	**3.0**	**73.0**	**7.0**	**21.0**	**15.3**	**100**
150	**35.3**	**24.0**	**7.0**	**139.0**	**12.0**	**42.5**	**32.3**	**100**
200	**59.5**	**43.0**	**15.0**	**235.0**	**31.0**	**82.5**	**54.8**	**100**
250	**112.2**	**96.0**	**5.0**	**353.0**	**67.5**	**131.0**	**97.6**	**100**
300	**193.3**	**141.0**	**15.0**	**741.0**	**95.0**	**254.5**	**161.6**	**100**
350	**355.4**	**215.0**	**7.0**	**1107.0**	**79.0**	**679.0**	**249.9**	**87**
400	**417.6**	**279.0**	**47.0**	**1245.0**	**81.0**	**719.0**	**300.3**	**77**
450	**312.4**	**345.0**	**41.0**	**649.0**	**126.5**	**473.5**	**258.6**	**50**
500	**476.6**	**543.0**	**61.0**	**1043.0**	**189.0**	**547.0**	**366.6**	**33**
50	**0.3**	**0.3**	**0.1**	**0.8**	**0.2**	**0.3**	**0.3**	**100**
100	**3.3**	**2.2**	**0.6**	**11.8**	**1.6**	**3.9**	**3.1**	**100**
150	**22.4**	**14.0**	**4.0**	**76.0**	9.4	28.3	**18.2**	**100**
200	**82.2**	**56.7**	**24.3**	**284.7**	**43.7**	**104.3**	**69.1**	**100**
250	**213.9**	**191.8**	30.8	**624.4**	**136.7**	**260.3**	**169.0**	**100**
300	**439.1**	**381.3**	**91.9**	**1387.5**	**224.9**	**521.4**	**360.3**	**100**
350	**1079.0**	720.5	**94.8**	**3339.2**	**324.2**	**1721.9**	**718.9**	**87**
400	**1421.9**	1497.6	**284.9**	**3090.9**	403.4	**2060.3**	**1040.5**	**77**
450	**1291.5**	1534.2	**301.1**	**2426.3**	**629.0**	**1838.2**	**1064.9**	**50**
500	2119.2	2200.6	**503.5**	**3396.8**	1265.3	3229.8	1730.6	**33**

2.4.4. The Impact of the Inclusion of Valid Inequalities

We tested different types of valid inequalities as described in Section 2.3.4. Unfortunately, incorporating the optimality cuts 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 of 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 depth-first search, and warmstarts are disabled. Let us quickly comment on why warmstarts are disabled for this test even though



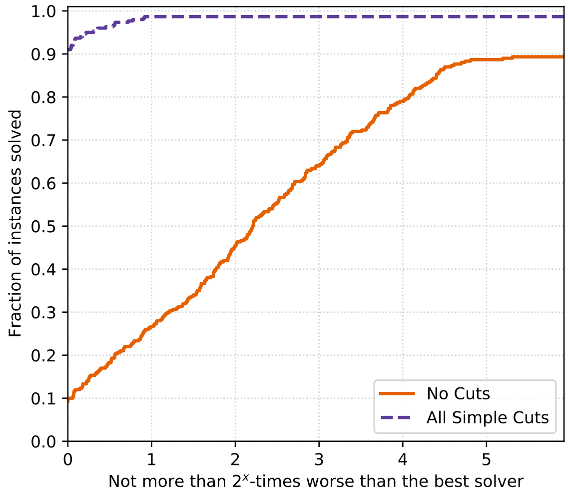


Figure 2.4.: Performance profiles for the number of branch-and-bounds nodes (left) and the running time (right) for variants with all possible simple cuts and without any.

they have a positive impact on the performance. For technical reasons, Gurobi needs both parameters VBasis and CBasis to warmstart a node problem, which contain the variable basis vector and the constraint basis vector. When cuts are added, the constraint basis vector needed to warmstart the problem is of higher dimension than the constraint basis vector of the parent node, which is why the use of VBasis/CBasis is mutually exclusive with the use of cuts. It would be possible, to use the parameters PStart/DStart, as Gurobi only needs a primal start pointing, which is available even with added cuts. However, as the difference between a warmstart with PStart/DStart and no warmstart is not significant, we choose to disable the warmstart here for simplicity. No instances are excluded for this test. As can be seen in Figure 2.4, incorporating the simple cuts has a great impact both on the number of branch-and-bound nodes as well as on the running time. This is also obvious from the results in Tables 2.4 and B.8. For almost all measures and instance sizes, the approach with the simple cuts significantly outperforms the method without the cuts both w.r.t. the node counts and the running times. Moreover, we see in Table 2.4 that we can solve almost all instances of the entire test set.

2.4.5. Comparison of the Method to Commercial Benchmark 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 and warmstarts are disabled. As mentioned before, we choose depth-first search as our node selection strategy.

In order to compare MILCP-PBB with other approaches from the literature, we consider the MILP Reformulation 2.1.5, that was proposed in Gabriel et al. (2013a). 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.

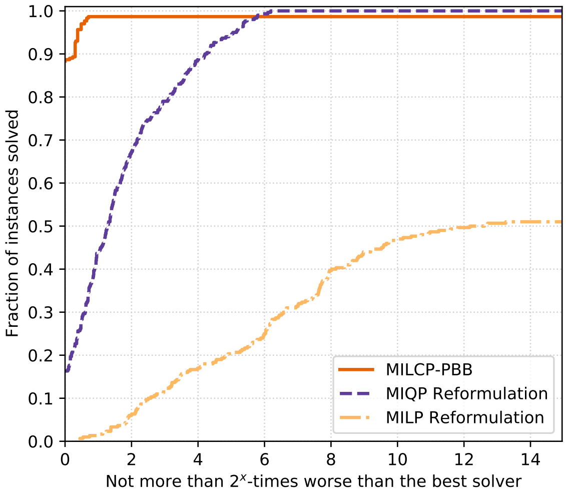
Table 2.4.: Aggregated node counts (top) and runtimes (bottom) for the valid inequalities test with all simple cuts

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
50	**7.7**	**7.0**	**3.0**	**15.0**	**5.0**	**11.0**	**7.7**	**100**
100	**8.6**	**9.0**	**3.0**	**15.0**	**5.5**	**11.0**	**8.6**	**100**
150	**12.4**	**13.0**	**7.0**	**23.0**	**9.5**	**13.0**	**12.3**	**100**
200	**15.8**	**16.0**	**9.0**	**27.0**	**13.0**	**17.0**	**15.7**	**100**
250	**18.5**	**19.0**	**5.0**	**29.0**	**17.0**	**21.0**	**18.4**	97
300	**23.0**	**23.0**	**11.0**	**37.0**	**19.5**	**27.0**	**22.9**	**100**
350	**30.0**	**31.0**	**7.0**	**47.0**	**21.0**	**37.0**	**29.7**	**97**
400	**37.3**	**33.0**	**17.0**	**81.0**	**22.0**	**50.0**	**36.4**	**100**
450	**32.4**	**28.0**	**23.0**	**63.0**	**27.0**	**31.0**	**32.1**	**93**
500	**35.4**	**35.0**	**23.0**	**51.0**	**32.0**	**37.0**	**35.2**	**100**
50	**0.2**	0.3	**0.1**	**0.3**	**0.2**	**0.3**	**0.2**	**100**
100	**1.2**	**1.2**	**0.7**	**2.2**	**1.1**	**1.3**	**1.2**	**100**
150	**4.5**	**4.4**	**2.8**	**6.7**	**3.9**	**4.9**	**4.4**	**100**
200	**10.8**	**10.7**	**7.9**	**14.5**	**9.5**	**11.9**	**10.7**	**100**
250	**20.5**	**20.6**	**13.5**	**26.9**	**17.2**	**23.4**	**20.2**	97
300	**34.5**	**32.9**	**27.0**	**45.6**	**29.8**	**39.2**	**34.1**	**100**
350	**57.2**	**56.1**	**33.1**	**80.4**	**48.8**	**64.4**	**56.2**	**97**
400	**85.0**	**78.8**	**61.9**	**150.2**	**69.6**	**92.6**	**82.9**	**100**
450	**104.6**	**101.6**	**84.9**	**167.8**	**91.4**	**112.8**	**103.2**	**93**
500	**124.9**	**128.3**	**74.5**	**155.0**	**116.5**	**134.3**	**123.0**	**100**

Furthermore, note that Problem 2.1.5 requires a significantly larger set of $3n + 2|I|$ variables. Besides the significantly larger number of variables, one additional drawback of Problem 2.1.5 is that it requires to determine sufficiently large big-B constraints. However, we can actually modify Problem 2.1.5 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 leads to the following reformulation.

Reformulation 2.4.1. The LCP(q, M, I) can be reformulated with $\alpha \in (0, 1)$ as:

$$
\begin{aligned}
\min_{z,z',\sigma} \quad & \alpha z^\top (q + Mz) + (1 - \alpha) \sum_{i \in I} \sigma_i && \text{(2.4.1a)}\\
\text{s.t.} \quad & z \geq 0, \quad q + Mz \geq 0, && \text{(2.4.1b)}\\
& 0 \leq z_I \leq z' + \sigma, && \text{(2.4.1c)}\\
& z' - \sigma \leq z_I \leq 1, && \text{(2.4.1d)}\\
& z \in \mathbb{R}^n, \quad z' \in \{0, 1\}^I, && \text{(2.4.1e)}\\
& \sigma \in \mathbb{R}^I_{\geq 0}. && \text{(2.4.1f)}
\end{aligned}
$$



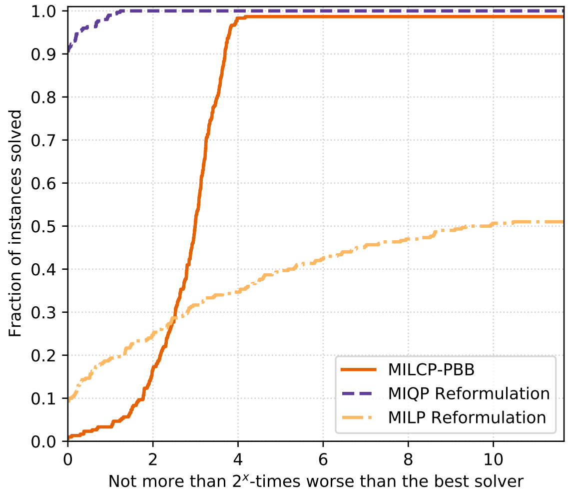


Figure 2.5.: Performance profiles for the number of branch-and-bound nodes (left) and the running time (right) for the MIQP reformulation, the MILP reformulation and MILCP-PBB.

Instead of using variables ρ_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 2.1.5 with z'_i being the corresponding indicator variables as before. Note that the MIQP 2.4.1 only has $|I|$ additional binary variables z' and $|I|$ additional continuous variables σ_i when compared to the original MILCP. Thus, the number of additionally required auxiliary variables is significantly reduced compared to the MILP reformulation 2.1.5. This makes a huge difference in practice: Gurobi is able to solve the MIQP 2.4.1 in significantly less time compared to what is required for solving the MILP 2.1.5; see Figure 2.5. When 2.1.5 and 2.4.1 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 and no instances are excluded. Figure 2.5 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 times. 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 inefficiencies from our Python implementation. More details can be found in Tables 2.5, 2.6, and B.9. It is evident that our approach clearly outperforms the two benchmark approaches w.r.t. the node count for all measures and sizes. For the running time it is evident that the MIQP approach outperforms both our approach and the MILP formulation.

As the MIQP reformulation has no failures and MILCP-PBB only has four, 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

Table 2.5.: Aggregated node counts (top) and runtimes (bottom) for the first benchmark test for MILCP-PBB

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
50	**7.7**	7.0	**3.0**	**15.0**	5.0	**11.0**	**7.7**	**100**
100	**8.6**	**9.0**	**3.0**	**15.0**	5.5	**11.0**	**8.6**	**100**
150	**12.4**	**13.0**	7.0	**23.0**	**9.5**	**13.0**	**12.3**	**100**
200	**15.8**	**16.0**	**9.0**	**27.0**	**13.0**	**17.0**	**15.7**	**100**
250	**19.3**	**19.0**	11.0	**29.0**	**17.0**	**21.5**	**19.2**	97
300	**26.5**	**26.0**	13.0	**37.0**	**23.5**	**33.0**	**26.3**	**100**
50	0.2	0.3	0.1	0.3	0.2	0.3	0.2	**100**
100	1.2	1.2	0.7	2.2	1.1	1.3	1.2	**100**
150	4.5	4.4	2.8	6.7	3.9	4.9	4.4	**100**
200	10.8	10.7	7.9	14.5	9.5	11.9	10.7	**100**
250	20.8	20.7	14.1	26.9	17.6	23.5	20.6	97
300	38.5	38.8	29.6	45.6	35.5	42.2	38.2	**100**

Table 2.6.: Aggregated node counts (top) and runtimes (bottom) for the first benchmark test for the MIQP reformulation

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
50	9.9	**6.0**	**3.0**	27.0	**4.0**	15.0	9.6	**100**
100	11.3	9.5	**3.0**	39.0	**5.2**	13.8	11.0	**100**
150	21.4	16.5	**5.0**	61.0	10.0	26.2	20.5	**100**
200	35.4	28.0	12.0	75.0	21.0	50.5	34.2	**100**
250	65.4	66.0	**7.0**	175.0	35.8	71.5	60.4	**100**
300	128.0	77.0	**8.0**	447.0	61.5	142.8	102.6	**100**
50	**0.1**	**0.1**	**0.1**	**0.2**	**0.1**	**0.2**	**0.1**	**100**
100	**0.3**	**0.3**	**0.1**	**0.4**	**0.2**	**0.3**	**0.3**	**100**
150	**0.7**	**0.7**	0.5	**1.0**	**0.6**	**0.8**	**0.7**	**100**
200	**1.4**	**1.4**	**1.2**	**1.7**	**1.4**	**1.5**	**1.4**	**100**
250	**2.2**	**2.2**	**1.7**	**2.9**	**2.0**	**2.4**	**2.2**	**100**
300	**3.5**	**3.0**	**2.5**	**6.0**	**3.0**	**3.6**	**3.4**	**100**

a second test set of 300 random instances as before but doubled both the instance sizes as well as the number of integer variables. For this test, we also tripled the time limit and now consider as failures only those instances that are not solved within 3 h. The comparison of the methods applied to these instances is shown in Figure 2.6 and Tables 2.7 and B.10, where we excluded 86 instances that no method solved. We can notice that, again, the number of nodes needed by MILCP-PBB is significantly smaller than the number of nodes

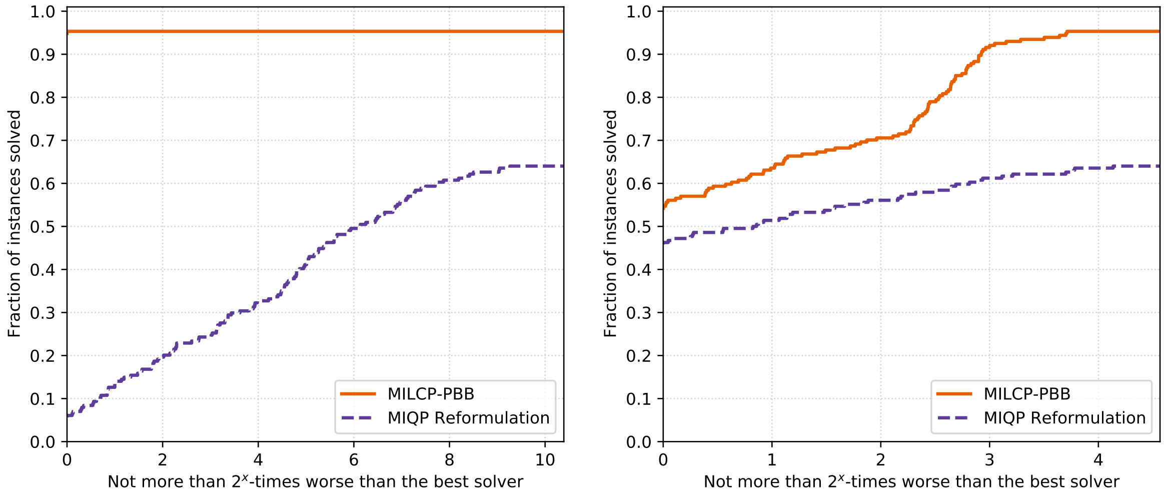


Figure 2.6.: Performance profiles for the number of branch-and-bound nodes (left) and the running time (right) for the MIQP reformulation and our algorithm (for the second test set with larger instances).

Table 2.7.: Aggregated node counts (top) and runtimes (bottom) for the second benchmark test for MILCP-PBB

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
100	**17.1**	**17.0**	**9.0**	**31.0**	**13.5**	**21.0**	**17.0**	**100**
200	**43.1**	**39.0**	**21.0**	**107.0**	**30.5**	**49.5**	**42.1**	93
300	**96.6**	**76.0**	**35.0**	**259.0**	**67.5**	**123.5**	**91.0**	93
400	**246.8**	**183.0**	**37.0**	**779.0**	**139.0**	**309.0**	**205.6**	87
500	**296.5**	**297.0**	**113.0**	**685.0**	**131.0**	**385.0**	**262.0**	**93**
600	**183.0**	**185.0**	**163.0**	**201.0**	**174.0**	**193.0**	**182.6**	**80**
100	2.4	2.4	1.7	3.1	2.2	2.6	2.4	**100**
200	21.7	19.5	13.4	52.9	17.3	24.6	21.0	93
300	99.6	92.9	49.0	**215.8**	70.9	123.2	93.5	93
400	**370.0**	**336.0**	134.3	**955.4**	**235.2**	**379.4**	**319.4**	87
500	**483.9**	**421.7**	**154.6**	**881.9**	**385.2**	**479.0**	**441.9**	**93**
600	**662.2**	**698.5**	**512.0**	**775.9**	**605.3**	**737.2**	**652.4**	**80**

needed by Gurobi. MILCP-PBB is also faster and has significantly less unsolved instances. Thus, it turns out to be more robust as well. MILCP-PBB and Gurobi have 19 as well as 49, respectively, failures on instances of Type (a), 34 and 53 failures on instances of Type (b), as well as 40 and 61 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 roughly 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 prevents it from proving optimality within the time limit. Extensive tables including node counts, running times, and optimality gaps for all instances including the instances not solved by both solvers can be found in Appendix C.

Chapter 3

Generalization of the Method

In this chapter, we want to extend the possibilities of the algorithm and give a proof-of-concept for problem classes that are very general. We discuss two possible generalizations. The first is a class of problems, that have a convex feasible set and an objective function in which the non-convex part is the sum of minimum functions, which we describe in Section 3.1. This class is the direct extension of the reformulation defined in Definition 2.1.6. The second class is the class of problems with a convex feasible set and an objective function in which the objective function is piecewise convex. We present this class in Section 3.2. While the first class we discuss is in fact a special case of the second class, its formulation is more compact and needs no additional information, which is why we discuss it separately. As these problem classes are purely of theoretical interest for now, we will not discuss a computational study.

3.1. Problems with a Sum of Minimum Functions

We want to describe a first very general class of problems, that could be solved with our method. In Section 3.1.1 we give a formal description of the problem class. In Section 3.1.2 we describe the generalization of our algorithm and show its correctness. In Section 3.1.3 we give first hints for possible enhancements of the method, such as valid inequalities.

3.1.1. Stating the Problem Class

In this section we will investigate non-convex, non-smooth optimization problems, whose objective functions are a convex function together with a sum of minimum functions each over a convex feasible set. The general version of this problem class reads as follows.

Definition 3.1.1. Let $h\colon \mathbb{R}^n \to \mathbb{R}$ and $g_{ij}\colon \mathbb{R}^n \to \mathbb{R},\ i \in I,\ j \in J_i$, be convex functions with I and $J_i,\ i \in I$, being index sets. Let $X \subseteq \mathbb{R}^n$ be a convex set. The optimization problem then reads

$$\min_{x\in\mathbb{R}^n} \quad h(x) + \sum_{i\in I} \min\{g_{ij}(x) \colon j \in J_i\} \tag{3.1.1a}$$

$$\text{s.t.} \quad x \in X. \tag{3.1.1b}$$

As seen before, the objective function is obviously not convex and therefore problematic to solve. The second part of the function $\sum_{i\in I} \min\{g_{ij}(x) \colon j \in J_i\}$ is non-convex in general, as the g_{ij} are convex and the minimum function is concave. This is the part of the problem, that we want to tackle by using our penalty branch-and-bound method, that we have described in the section before. In order to prove the correctness of the method, we need to assume, that the problem is bounded. We further need to assume, that the single parts of the second part of the objective function are bounded, i.e., $L_i := \min_{x\in X} \min\{g_{ij}(x) \colon j \in J_i\}$ for all $i \in I$ should be finite.

Lemma 3.1.2. *Let L_i as defined above be finite numbers for an instance of Problem 3.1.1 and let there be $i_1 \in I$, $j_1 \in J_i$, $x_1 \in X$ such that $g_{i_1 j_1}(x_1) < 0$ for that instance. We can find an equivalent formulation for Problem 3.1.1 with functions h', g'_{ij}, for which $g'_{ij}(x) \geq 0$ for all $i \in I$, $j \in J_i$, $x \in X$.*

Proof. Let $L := min_{i\in I} L_i$. We can then do the following transformation:

$$\begin{aligned}
& h(x) + \sum_{i\in I} \min\{g_{ij}(x) \colon j \in J_i\} \\
={}& h(x) - |I|L + \sum_{i\in I} (\min\{g_{ij}(x) \colon j \in J_i\} + L) \\
={}& h(x) - |I|L + \sum_{i\in I} \min\{g_{ij}(x) + L \colon j \in J_i\} \\
={}& h'(x) + \sum_{i\in I} \min\{g'_{ij}(x) \colon j \in J_i\}
\end{aligned}$$

with $h'(x) = h(x) - |I|L$ and $g'_{ij}(x) + L$, and the lemma follows. $\square$

Therefore, it is reasonable to assume in the following, that $g_{ij}(x) \geq 0$ for all $i \in I$, $j \in J_i$, $x \in X$. In the following, we will denote the objective function of Problem 3.1.1 by f, i.e.,

$$f(x) := h(x) + \sum_{i\in I} \min\{g_{ij}(x) \colon j \in J_i\}.$$

3.1.2. The Penalty Branch-and-Bound Method for this Class

In this section we will give a generalized version of the algorithm presented in Section 2.2. The algorithm we propose solves the non-convex optimization problem described in Definition 3.1.1 to global optimality.

Branching

Analogously to Section 2.2, we obtain the root node by ignoring the non-convex part of the objective function. Therefore the root node is the following convex optimization problem:

Definition 3.1.3 (Root Node Problem of PBB for Problem 3.1.1)**.** The root node problem of the PBB for Problem 3.1.1 is defined as

$$\min_{x\in\mathbb{R}^n} \quad h(x) \tag{3.1.2a}$$

$$\text{s.t.} \quad x \in X. \tag{3.1.2b}$$

By definition, this optimization problem is convex and therefore tractable. We then add child nodes to the branching tree, for which the objective function is altered by adding penalty terms. Here, the number of child nodes is not necessarily two but depends on the branching index $i \in I$ and the size of the index set J_i. For every $j \in J_i$ we add a child node, where we add the term g_{ij} to the objective function. The child nodes of the root node therefore read as follows:

Definition 3.1.4 (Child Problems of the Root Node of PBB for Problem 3.1.1)**.** The set of child nodes of the root node of PBB for Problem 3.1.1 are defined as the problems

$$\min_{x\in\mathbb{R}^n} \quad h(x) + g_{ij}(x) \tag{3.1.3a}$$

$$\text{s.t.} \quad x \in X, \tag{3.1.3b}$$

for all $j \in J_i$ with $i \in I$ being the chosen branching index.

Analogously to Section 2.2 the idea is to compute the minimum of the minimum function $\min\{g_{ij}(x) \colon j \in J_i\}$ over $x \in X$, by taking the minimum of the single functions, i.e.,

$$\min_{x\in X}\left\{\min_{j\in J_i} g_{ij}(x)\right\} = \min_{j\in J_i}\left\{\min_{x\in X} g_{ij}(x)\right\}.$$

We then solve the child nodes independently as we did before and repeat the process. As with every branching decision we keep adding convex functions, every node of the branch-and-bound tree is a convex optimization problem. An arbitrary node N in the tree can be described by the branching decisions that have been made. We define $I_N \subseteq I$ to be the indices for which branching decisions have been made up to node N. We further define J_N to be the $|I_N|$-dimensional tuple containing all branching decisions. In other words, $J_N := \left(j_1, \ldots, j_{|I_N|}\right)$ with $(J_N)_i$ being the branching decision for index $(I_N)_i$. We can then describe any node N by these vectors. In the following we will also denote the branching decision taken for an index $i \in I$ by $j_i \in J_i$. Then, the problem at node $N = (I_N, J_N)$ reads as follows.

Definition 3.1.5 (Node Problem of PBB for Problem 3.1.1)**.** The node problem of PBB for Problem 3.1.1 at node $N = (I_N, J_N)$ is defined as

$$\min_{x \in \mathbb{R}^n} \quad h(x) + \sum_{i \in I_N} g_{ij_i}(x) \tag{3.1.4a}$$

$$\text{s.t.} \quad x \in X. \tag{3.1.4b}$$

We denote the optimal solution of that problem by x_N^* and the objective function by f_N, i.e.,

$$f_N(x) := h(x) + \sum_{i \in I_N} g_{ij_i}(x).$$

Analogously to Section 2.2, we first show that this branching routine would solve Problem 3.1.1 by iterating over all possible combinations.

Lemma 3.1.6. *Let x^* be an optimal solution of Problem 3.1.1. Then, it holds*

$$f(x^*) = \min \left\{ f_N(x_N^*) \colon N = (I_N, J_N) \text{ with } I_N = I \right\}.$$

Proof. Note that the feasible set does not depend on N. Hence, all optimal points are feasible for all nodes. Let $N^* = (I_{N^*}, J_{N^*})$ with $I_{N^*} = I$ and $J_{N^*} = \left(j^*_{(I_{N^*})_1}, \dots, j^*_{(I_{N^*})_{|I_{N^*}|}} \right)$ be the leaf with $g_{ij_i^*}(x) \le g_{ij}(x)$ for all $j \in J_i$ and $i \in I$. We then have

$$\begin{aligned} f(x^*) &= h(x^*) + \sum_{i \in I} \min\{g_{ij}(x^*) \colon j \in J_i\} \\ &= h(x^*) + \sum_{i \in I} g_{ij_i^*}(x^*) \\ &= f_{N^*}(x^*) \ge f_{N^*}(x_{N^*}^*). \end{aligned}$$

Hence,

$$f(x^*) \ge \min \left\{ f_N(x_N^*) \colon N = (I_N, J_N) \text{ with } I_N = I \right\}$$

holds. To show the other inequality, we assume that there exists a node $N' = (I_{N'}, J_{N'})$ with $I_{N'}$ and $J_{N'} = (j'_1, \dots, j'_{|I_{N'}|})$ such that

$$f_{N'}(x_{N'}^*) < f(x^*)$$

holds. We thus obtain $f_{N'}(x_{N'}^*) < f(x_{N'}^*)$ or, equivalently,

$$h(x_{N'}^*) + \sum_{i \in I} g_{ij'_i}(x_{N'}^*) < h(x_{N'}^*) + \sum_{i \in I} \min\{g_{ij}(x_{N'}^*) \colon j \in J_i\}.$$

This implies

$$\sum_{i \in I} \left(g_{ij'_i}(x_{N'}^*) - \min\{g_{ij}(x_{N'}^*) \colon j \in J_i\} \right) < 0,$$

which is impossible as

$$g_{ij'_i}(x^*_{N'}) \geq \min\{g_{ij}(x^*_{N'}) \colon j \in J_i\}.$$

Hence,

$$f(x^*) \leq \min\{f_N(x^*_N) \colon N = (I_N, J_N) \text{ with } I_N = I\}$$

holds and the claim follows. □

Bounding

We now want to show, that it is not necessary to compute solutions for all possible combinations of functions but that, instead, we can establish upper and lower bounds to prune certain branches of the branch-and-bound tree as we did in Section 2.2 and as done in classic branch-and-bound methods. Again, as the feasible set does not change throughout the process, every optimal solution of every node in the tree yields a global upper bound when plugged into the objective function of the master problem. We denote x^*_{inc} to be the incumbent of the process, i.e., the point so that $f(x^*_{\text{inc}})$ constitutes the best known upper bound. We now want to show, that we can also establish local lower bounds.

Lemma 3.1.7. *Let $N' = (I_{N'}, J_{N'})$ be a successor of some node $N = (I_N, J_N)$ in the branch-and-bound tree, i.e., $I_N \subseteq I_{N'}$ and $(J_N)_i = (J_{N'})_i$ for all $i \in I_N$ holds. Then,*

$$f_N(x^*_N) \leq f_{N'}(x^*_{N'})$$

holds.

Proof. Since the feasible set does not change during the branching process all feasible points remain feasible for all nodes. Thus,

$$\begin{aligned}
f_{N'}(x^*_{N'}) &= h(x^*_{N'}) + \sum_{i \in I_{N'}} g_{ij'_i}(x^*_{N'}) \\
&= h(x^*_{N'}) + \sum_{i \in I_N} g_{ij'_i}(x^*_{N'}) + \sum_{i \in I_{N'} \setminus I_N} g_{ij'_i}(x^*_{N'}) \\
&\geq h(x^*_{N'}) + \sum_{i \in I_N} g_{ij'_i}(x^*_{N'}) \\
&= h(x^*_{N'}) + \sum_{i \in I_N} g_{ij_i}(x^*_{N'}) \\
&= f_N(x^*_{N'}) \geq f_N(x^*_N).
\end{aligned}$$

Note that the first inequality is due to the fact that $g_{ij'_i}(x^*_{N'}) \geq 0$ for $i \in I_N$ on the feasible set. The second inequality follows from optimality. □

From Lemma 3.1.7 we therefore know that if it is the case in some node N during the process that

$$f_N(x^*_N) \geq f(x^*_{\text{inc}}),$$

we know that any leaf node of the subtree rooted in N will not yield a better solution than the incumbent we already know does. Hence, we can prune that subtree. Note, that in general we cannot prune due to "feasibility", as we are not looking for solutions in the sense of Chapter 2.

Algorithmic Description

We are now ready to give an algorithmic description of the method in Algorithm 3 and show its correctness.

Algorithm 3 A Penalty Branch-and-Bound Algorithm for Problem 3.1.1

Input: $h\colon \mathbb{R}^n \to \mathbb{R}$, $g_{ij}\colon \mathbb{R}^n \to \mathbb{R}$, $i \in I$, $j \in J_i$ convex functions, I, J_i, $i \in I$, index sets, $X \subseteq \mathbb{R}^n$
Output: A global optimum x^* of Problem 3.1.1.
Set $\mathcal{N} \leftarrow \{(\emptyset, \emptyset)\}$, $f_{\text{inc}} \leftarrow \infty$, and $x^*_{\text{inc}} \leftarrow \mathsf{none}$.
while $\mathcal{N} \neq \emptyset$ **do**
  Choose $N = (I_N, J_N) \in \mathcal{N}$ and set $\mathcal{N} \leftarrow \mathcal{N} \setminus \{N\}$.
  Compute $x^*_N \in \operatorname{argmin}\{f_N(x) : x \in X\}$.
  if $f(x^*_N) < f_{\text{inc}}$ **do**
    Set $x^*_{\text{inc}} \leftarrow x^*_N$ and $f_{\text{inc}} \leftarrow f(x^*_N)$.
  if $f_N(x^*_N) < f_{\text{inc}}$ **and** $I \setminus I_N \neq \emptyset$ **do**
    Choose $i \in I \setminus I_N$, set $\mathcal{N} \leftarrow \mathcal{N} \cup \bigcup_{j \in J_i} (I_N \cup \{i\}, J_N \times \{j\})$.
return x^*_{inc}

Theorem 3.1.8. *Algorithm 3 terminates after finitely many steps with a global optimal solution of Problem 3.1.1, if the root node problem is bounded and L_i for all $i \in I$ are finite.*

Proof. The algorithm terminates after finitely many steps since the set I is finite. Thus, at some point, $I_N = I$ holds and we can no longer find a branching variable in the node and no child node can be generated. We only add a finite number of nodes in every step, as the sets J_i are finite. Assume now that $f_N(x^*_N) < f(x^*_{\text{inc}})$ always holds in the second if-clause. Then the correctness of the algorithm follows from Lemma 3.1.6, as we iterate through the complete branch-and-bound tree. Finally, in the cases, in which $f_N(x^*_N) \geq f(x^*_{\text{inc}})$ holds, the nodes that are not added can be excluded due to Lemma 3.1.7. □

While the correctness of the algorithm and convexity of the different node problems does not depend on the structure of the index sets I and J_i, it is to expect that they have an impact on the performance of the algorithm. For example, if the set I only has one element and the set J_1 is very large, the algorithm will enumerate over all possibilities and the bounding part will never come into effect. Therefore it is to be expected, that for a large cardinality of the set$I$ and small cardinalities for the setsJ_i, as it is the case in Section 2.2, the strengths of the algorithm are used best.

3.1.3. Further Enhancements of the Method

We want to consider some improvements of the method above. Again, the choice of the next node to solve and the choice of the next index to branch on are not specified. But we will not investigate these choices, as strategies for both choices are heuristics that have to be evaluated in numerical experiments. However, we want to show that a generalization of the simple cuts as introduced in Section 2.3.4 is possible here as well.

For the simple cuts in Section 2.3.4 we have exploited that the break point for which the penalty term z_j gets bigger than the term $(1 - z_j)$ is known. In the generalized version of the cut, we do not know this point but instead create inequalities that pairwise compare the different penalty terms. So if we have branched on index $i \in I$ by adding the penalty function $g_{ij_i}(x)$ with $j_i \in J_i$, we know that we can enforce that penalization to be smaller than the penalizations by the other terms $g_{ij}(x)$ for all $j \neq j_i$. In other words, at every node $N = (I_N, J_N)$ we can add

$$g_{ij_i}(x) \leq g_{ij}(x) \text{ for all } i \in I \text{ and } j \in J_i.$$

This is obviously only a sensible thing to do when these inequalities describe a convex set, i.e., when the functions g_{ij} are linear. We will again show first, that the optimal solution will not be cut-off during the process. For this section, we will denote the optimal solution of the modified node problem by x_N^*, i.e.,

$$x_N^* \in \operatorname{argmin} \left\{ f_N(x) \colon x \in X,\ g_{ij_i}(x) \leq g_{ij}(x) \text{ for all } i \in I,\ j \in J_i \right\}.$$

Lemma 3.1.9. *Let x_N^* be an optimal solution at node $N = (I_N, J_N)$ when all cuts are included. Then,*

$$f(x^*) = \min \left\{ f_N(x_N^*) \colon N = (I_N, J_N) \textit{ with } I_N = I \right\}$$

holds.

Proof. Let $N^* = (I_{N^*}, J_{N^*})$ with $I_{N^*} = I$ and $J_{N^*} = \left(j_1^*, \ldots, j_{|I_{N^*}|}^* \right)$ be the leaf with $g_{ij_i^*}(x) \leq g_{ij}(x)$ for all $j \in J_i$ and $i \in I$. We then have

$$\begin{aligned} f(x^*) &= h(x^*) + \sum_{i \in I} \min\{g_{ij}(x^*) \colon j \in J_i\} \\ &= h(x^*) + \sum_{i \in I} g_{ij_i^*}(x^*) \\ &= f_{N^*}(x^*) \geq f_{N^*}(x_{N^*}^*). \end{aligned}$$

The last inequality holds, because by definition x^* does not violate any of the simple cuts added and is therefore feasible for node N^*. Hence,

$$f(x^*) \geq \min \left\{ f_N(x_N^*) \colon N = (I_N, J_N) \text{ with } I_N = I \right\}$$

holds. To show the other inequality, we assume that there exists a node $N' = (I_{N'}, J_{N'})$ with $I_{N'} = I$ and $J_{N'} = (j'_1, \dots, j'_{|I_{N'}|})$ such that

$$f_{N'}(x^*_{N'}) < f(x^*)$$

holds. We thus obtain $f_{N'}(x^*_{N'}) < f(x^*_{N'})$ or, equivalently,

$$h(x^*_{N'}) + \sum_{i \in I} g_{ij'_i}(x^*_{N'}) < h(x^*_{N'}) + \sum_{i \in I} \min\{g_{ij}(x^*_{N'}) \colon j \in J_i\}.$$

This implies

$$\sum_{i \in I} \Big(g_{ij'_i}(x^*_{N'}) - \min\{g_{ij}(x^*_{N'}) \colon j \in J_i\} \Big) < 0,$$

which is impossible as

$$g_{ij'_i}(x^*_{N'}) \geq \min\{g_{ij}(x^*_{N'}) \colon j \in J_i\}.$$

Hence,

$$f(x^*) \leq \min\{f_N(x^*_N) \colon N = (I_N, J_N) \text{ with } I_N = I\}$$

holds and the claim follows. □

Next we show that the bounding step of the algorithm remains correct as well when all cuts are added.

Lemma 3.1.10. *Let $N' = (I_{N'}, J_{N'})$ be a successor of some node $N = (I_N, J_N)$ in the branch-and-bound tree, i.e., $I_N \subseteq I_{N'}$ and $(J_N)_i = (J_{N'})_i$ for all $i \in I_N$ holds. Then,*

$$f_N(x^*_N) \leq f_{N'}(x^*_{N'})$$

holds.

Proof. By definition, we have

$$\begin{aligned}
f_{N'}(x^*_{N'}) &= h(x^*_{N'}) + \sum_{i \in I_{N'}} g_{ij'_i}(x^*_{N'}) \\
&= h(x^*_{N'}) + \sum_{i \in I_N} g_{ij'_i}(x^*_{N'}) + \sum_{i \in I_{N'} \setminus I_N} g_{ij'_i}(x^*_{N'}) \\
&\geq h(x^*_{N'}) + \sum_{i \in I_N} g_{ij'_i}(x^*_{N'}) \\
&= h(x^*_{N'}) + \sum_{i \in I_N} g_{ij_i}(x^*_{N'}) \\
&= f_N(x^*_{N'}) \geq f_N(x^*_N).
\end{aligned}$$

Note that the first inequality is due to the fact that $g_{ij'_i}(x^*_{N'}) \geq 0$ for all $i \in I_N$ on the feasible set. The second inequality follows from optimality as the feasible sets are nested in the sense that if a point is feasible for node N, it is also feasible for any successor node N'. □

Theorem 3.1.11. *Algorithm 3 remains correct if simple cuts*

$$g_{ij_i}(x) \le g_{ij}(x) \text{ for all } i \in I \text{ and } j \in J_i$$

are added at any node $N = (I_N, J_N)$.

Proof. From Lemma 3.1.9, we know that the optimal solution of Problem 3.1.1 is the optimal solution of a leaf node. From Lemma 3.1.10, 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 x^*_{inc} of Problem 3.1.1 and some node N for which

$$f(x^*_{\text{inc}}) \le f_N(x^*_N)$$

holds, we know that x^*_{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 3 remains correct when simple cuts are used. □

3.2. Problems with a Sum of Piecewise Convex Functions

We now come to a second class of problems that can be solved with our method. In Section 3.2.1 we give a formal description of the problem class. In Section 3.2.2 we describe another general form of our algorithm and show its correctness.

3.2.1. Stating the Problem Class

In this section we will investigate a different generalization for which we can use our penalty branch-and-bound method. It is a different class of non-convex, non-smooth optimization problems, whose objective functions are piecewise convex. While in general this generalization covers a bigger set of objective functions, we need additional information. In order for the algorithm to work, we need to know the breakpoints in the objective functions, i.e., the sets of the domain for which the objective function is convex. The problem class then reads as follows.

Definition 3.2.1. Let $h\colon \mathbb{R}^n \to \mathbb{R}$ and $g_{ij}\colon A_{ij} \to \mathbb{R},\ i \in I,\ j \in J_i$ be convex functions with I and $J_i,\ i \in I$, being index sets and $A_{ij} \subseteq \mathbb{R}^n$ being compact sets for all $j \in J_i, i \in I$ and with their interior points being pairwise disjoint for $j \in J_i$ and a fixed $i \in I$. Let $X \subseteq \mathbb{R}^n$ be a convex set. The optimization problem then reads

$$\min_{x \in \mathbb{R}^n} \quad h(x) + \sum_{i \in I} \sum_{j \in J_i} g_{ij}(x) \chi_{A_{ij}}(x) \tag{3.2.1a}$$

$$\text{s.t.} \quad x \in X, \tag{3.2.1b}$$

where $\chi_{A_{ij}}(x)$ is the characteristic function of the set A_{ij}, i.e.,

$$\chi_{A_{ij}}(x) := \begin{cases} 1, & \text{if } x \in A_{ij}, \\ 0, & \text{else.} \end{cases}$$

Again, the objective function is obviously non-convex. The second part of the function $\sum_{i \in I} \sum_{j \in J_i} g_{ij}(x)\chi_{A_{ij}}(x)$ is non-convex in general. This is again the part of the problem, that we want to tackle by using our penalty branch-and-bound method, that we have described before. As the A_{ij} are compact, we can assume that $L_i := \min_{x \in X} \sum_{j \in J_i} g_{ij}(x)\chi_{A_{ij}}(x)$ is finite for all $i \in I$. With the same arguments as before, we can therefore assume in the following, that $g_{ij}(x) \geq 0$ for all $i \in I, j \in J_i, x \in X$. In the following, we will denote the objective function of Problem 3.2.1 by f, i.e.,

$$f(x) := h(x) + \sum_{i \in I} \sum_{j \in J_i} g_{ij}(x)\chi_{A_{ij}}(x).$$

3.2.2. The Penalty Branch-and-Bound Method for this Class

In this section we will give another generalized version of the algorithm presented in Section 2.2. The algorithm we propose solves the non-convex optimization problem described in Definition 3.2.1 to global optimality.

Branching

Analogously to Sections 2.2 and 3.1.2, we obtain the root node by ignoring the non-convex part of the objective function. Therefore the root node is the following convex optimization problem:

Definition 3.2.2 (Root Node Problem of PBB for Problem 3.2.1)**.** The root node problem of the PBB for Problem 3.2.1 is defined as

$$\min_{x \in \mathbb{R}^n} \quad h(x) \tag{3.2.2a}$$

$$\text{s.t.} \quad x \in X. \tag{3.2.2b}$$

Again, we then add child nodes to the branching tree, for which the objective function is extended by adding penalty terms. Again, the number of child nodes is not necessarily two but depends on the branching index $i \in I$ and the size of the index set J_i. As a big difference to the other generalization, it is necessary to change the feasible set, as we need to make sure, that for each point of the feasible node the objective function coincides with the objective function of the master problem. The child nodes therefore are the following set of nodes:

Definition 3.2.3 (Child Problems of the Root Node of PBB for Problem 3.2.1)**.** The set of child nodes of the root node of PBB for Problem 3.2.1 are defined as the problems

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & h(x) + g_{ij}(x) && \text{(3.2.3a)} \\ \text{s.t.} \quad & x \in X, && \text{(3.2.3b)} \\ & x \in A_{ij}, && \text{(3.2.3c)} \end{aligned}$$

for all $j \in J_i$ with $i \in I$ being the chosen branching index.

The rationale is the same as before. We want to compute the minimum of the sum $\sum_{j \in J_i} g_{ij}(x)\chi_{A_{ij}}(x)$ over $x \in X$, by taking the minimum of the single functions on their respective domains, i.e.,

$$\min_{x \in X} \left\{ \sum_{j \in J_i} g_{ij}(x)\chi_{A_{ij}}(x) \right\} = \min_{j \in J_i} \left\{ \min_{x \in X \cap A_{ij}} g_{ij}(x)\chi_{A_{ij}}(x) \right\}.$$

We then solve the child nodes independently as we did before and repeat the process. As with every branching decision we keep adding convex functions and convex constraints to the feasible set, every node of the branch-and-bound tree is a convex optimization problem. An arbitrary node N in the tree can be described by the branching decisions that have been made. We denote the nodes the same way as we did in Section 3.1 and define $I_N \subseteq I$ to be the indices for which branching decisions have been made at node N. We further define J_N to be the $|I_N|$-dimensional tuple containing all branching decisions. In other words, $J_N := \left(j_1, \ldots, j_{|I_N|}\right)$ with $(J_N)_i$ being the branching decision for index $(I_N)_i$. Again, we can define w.l.o.g. that $j_i \in J_i$ describes the corresponding branching decision for $i \in I$. We can then describe any node N by these objects. Then, the problem at node $N = (I_N, J_N)$ reads as follows.

Definition 3.2.4 (Node Problem of PBB for Problem 3.2.1)**.** The node problem of PBB for Problem 3.2.1 at node $N = (I_N, J_N)$ is defined as

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & h(x) + \sum_{i \in I_N} g_{ij_i}(x) && \text{(3.2.4a)} \\ \text{s.t.} \quad & x \in X, && \text{(3.2.4b)} \\ & x \in \bigcap_{i \in I_N} A_{ij_i}. && \text{(3.2.4c)} \end{aligned}$$

We denote the optimal solution of that problem by x_N^* and the objective function by f_N, i.e.,

$$f_N(x) := h(x) + \sum_{i \in I_N} g_{ij_i}(x).$$

Again, we first show that this branching routine would solve Problem 3.2.1 by iterating through all possible combinations.

Lemma 3.2.5. *Let x^* be an optimal solution of Problem 3.2.1. Then, it holds*

$$f(x^*) = \min \{f_N(x_N^*) \colon N = (I_N, J_N) \text{ with } I_N = I\}.$$

Proof. Let $N^* = (I_{N^*}, J_{N^*})$ with $I_{N^*} = I$ and $J_{N^*} = \left(j_1^*, \ldots, j_{|I_{N^*}|}^*\right)$ be the leaf with $x^* \in A_{ij}$ for all $j \in J_i$ and $i \in I$. We then have

$$\begin{aligned} f(x^*) &= h(x^*) + \sum_{i \in I} \sum_{j \in J_i} g_{ij}(x) \chi_{A_{ij}}(x) \\ &= h(x^*) + \sum_{i \in I} g_{ij_i^*}(x^*) \\ &= f_{N^*}(x^*) \geq f_{N^*}(x_{N^*}^*). \end{aligned}$$

The last inequality holds due to optimality as the point x^* is feasible for node N^*. Hence,

$$f(x^*) \geq \min \{f_N(x_N^*) \colon N = (I_N, J_N) \text{ with } I_N = I\}$$

holds. To show the other inequality, we assume that there exists a node $N' = (I_{N'}, J_{N'})$ with $I_{N'} = I$ and $J_{N'} = \left(j_1', \ldots, j_{|I_{N'}|}'\right)$ such that

$$f_{N'}(x_{N'}^*) < f(x^*)$$

holds. We thus obtain $f_{N'}(x_{N'}^*) < f(x_{N'}^*)$ or, equivalently,

$$h(x_{N'}^*) + \sum_{i \in I} g_{ij_i'}(x_{N'}^*) < h(x_{N'}^*) + \sum_{i \in I} \sum_{j \in J_i} g_{ij}(x_{N'}^*) \chi_{A_{ij}}(x_{N'}^*).$$

Because the $g_{ij}(x)$ are non-negative on the feasible set, we therefore know that there is an index $\hat{\imath} \in I$ for which

$$g_{\hat{\imath} j_{\hat{\imath}}'}(x_{N'}^*) < \sum_{j \in J_i} g_{\hat{\imath} j}(x_{N'}^*) \chi_{A_{ij}}(x_{N'}^*).$$

By definition we know that $\chi_{A_{ij}}(x_{N'}^*) = 0$ for all $j \neq j_i'$ and $\chi_{A_{ij_i'}}(x_{N'}^*) = 1$. This implies

$$g_{\hat{\imath} j_{\hat{\imath}}'}(x_{N'}^*) < g_{\hat{\imath} j_{\hat{\imath}}'}(x_{N'}^*),$$

which is a contradiction. Hence,

$$f(x^*) \leq \min \{f_N(x_N^*) \colon N = (I_N, J_N) \text{ with } I_N = I\}$$

holds and the claim follows. □

Bounding

We now want to show, that it is not necessary to compute solutions for all possible combinations of functions, but that instead we can establish upper and lower bounds to prune certain branches of the branch-and-bound tree. As the feasible set of the master problem contains the feasible sets of all node problems, every optimal solution of every node in the tree yields a global upper bound when plugged into the objective function of the master problem. We denote x^*_{inc} to be the incumbent of the process, i.e., the point so that $f(x^*_{\text{inc}})$ constitutes the best known upper bound. We now want to show, that we can also establish local lower bounds.

Lemma 3.2.6. *Let $N' = (I_{N'}, J_{N'})$ be a successor of some node $N = (I_N, J_N)$ in the branch-and-bound tree, i.e., $I_N \subseteq I_{N'}$ and $(J_N)_i = (J_{N'})_i$ for all $i \in I_N$ holds. Then,*

$$f_N(x^*_N) \le f_{N'}(x^*_{N'})$$

holds.

Proof. By definition, we have

$$\begin{aligned} f_{N'}(x^*_{N'}) &= h(x^*_{N'}) + \sum_{i \in I_{N'}} g_{ij'_i}(x^*_{N'}) \\ &= h(x^*_{N'}) + \sum_{i \in I_N} g_{ij'_i}(x^*_{N'}) + \sum_{i \in I_{N'} \setminus I_N} g_{ij'_i}(x^*_{N'}) \\ &\ge h(x^*_{N'}) + \sum_{i \in I_N} g_{ij'_i}(x^*_{N'}) \\ &= h(x^*_{N'}) + \sum_{i \in I_N} g_{ij_i}(x^*_{N'}) \\ &= f_N(x^*_{N'}) \ge f_N(z^*_N). \end{aligned}$$

Note that the first inequality is due to the fact that $g_{ij'_i}(x^*_{N'}) \ge 0$ for all $i \in I_N$ on the feasible set. The second inequality follows from optimality, as the feasible sets are nested in the sense, that if a point is feasible for node N, it is also feasible for any successor node N'. □

From Lemma 3.2.6 we therefore know, that if during the process it is the case in some node N, that

$$f_N(x^*_N) \ge f(x^*_{\text{inc}}),$$

we know that any leaf node of the subtree rooted in N will not yield a better solution than the incumbent we already know does. Hence, we can prune that subtree. Note, that again we cannot prune due to "feasibility" in general, as we are not looking for feasible points in the sense of Chapter 2. In this version of a penalty branch-and-bound method, we have to add additional constraints to the constraint set with every branching decision, similar to classic branch-and-bound algorithms. Therefore, it is possible, that some node problems become infeasible and we can make use of the pruning due to infeasibility as known from classic branch-and-bound methods.

Algorithmic Description

We are now ready to give an algorithmic description of the method in Algorithm 4 and show its correctness.

Algorithm 4 A Penalty Branch-and-Bound Algorithm for Problem 3.2.1

Input: $h\colon \mathbb{R}^n \to \mathbb{R}$, $g_{ij}\colon A_{ij} \to \mathbb{R}$, $i \in I$, $j \in J_i$, convex functions, A_{ij} compact sets, I, J_i, $i \in I$ index sets, $X \subseteq \mathbb{R}^n$
Output: A global optimum x^* of Problem 3.2.1.
Set $\mathcal{N} \leftarrow \{(\emptyset, \emptyset)\}$, $f_{\text{inc}} \leftarrow \infty$, and $x^*_{\text{inc}} \leftarrow \textsf{none}$.
while $\mathcal{N} \neq \emptyset$ **do**
  Choose $N = (I_N, J_N) \in \mathcal{N}$ and set $\mathcal{N} \leftarrow \mathcal{N} \setminus \{N\}$.
  Compute $x^*_N \in \operatorname{argmin}\{f_N(x) : x \in X,\ x \in \bigcap_{i \in I_N} A_{ij_i}\}$.
  if x^*_N exists and $f(x^*_N) < f_{\text{inc}}$ **do**
    Set $x^*_{\text{inc}} \leftarrow x^*_N$ and $f_{\text{inc}} \leftarrow f(x^*_N)$.
  if $f_N(x^*_N) < f_{\text{inc}}$ **and** $I \setminus I_N \neq \emptyset$ **do**
    Choose $i \in I \setminus I_N$, set $\mathcal{N} \leftarrow \mathcal{N} \cup \bigcup_{j \in J_i} \{(I_N \cup \{i\}, J_N \times j)\}$.
return x^*_{inc}

Theorem 3.2.7. *Algorithm 4 terminates after finitely many steps with a global optimal solution of Problem 3.2.1, if the root node problem is bounded.*

Proof. The algorithm terminates after finitely many steps since the set I is finite. Thus, at some point, $I = I_N$ holds and we can no longer find a branching variable in the node and no child node can be generated. We only add finite number of nodes in every step, as the sets J_i are finite. Assume now that $f_N(x^*_N) < f(x^*_{\text{inc}})$ always holds in the second if-clause. Then the correctness of the algorithm follows from Lemma 3.2.5, as we iterate through the complete branch-and-bound tree. Finally, in the cases, in which $f_N(x^*_N) \geq f(x^*_{\text{inc}})$ holds, the nodes that are not added can be excluded due to Lemma 3.2.6. □

Note, that the cutting planes presented as enhancements in Section 3.1.3 for Algorithm 3 are necessarily included in the algorithm for this generalization. Again, while the correctness of the algorithm and the convexity of the different node problems does not depend on the structure of the index sets I and J_i, it is to expect that they have an impact on the performance of the algorithm. For example, if the set I only has one element and the resulting set J_1 is very large, the algorithm will enumerate over all possibilities and the bounding part will never come into effect. Therefore it is to be expected, that a large cardinality of the set I and small cardinalities for the sets J_i, as it is the case in Section 2.2, the strength of the algorithm are used best.

Chapter 4

Solving Non-Monotone Mixed-Integer Linear Complementarity Problems

The canonical extension of monotone mixed-integer linear complementarity problems, i.e., MILCPs with a positive semi-definite matrix M, are MILCPs, where the matrix M does not necessarily have this property. For positive semi-definite matrices, the node problems given in Definition 2.2.4 are tractable problems, as the objective function is convex and quadratic. In the general case, we cannot assume convexity, hence making it not sensible to base our approach on Reformulation 2.1.6 to solve non-monotone MILCP. Therefore, we need to find a different formulation, which we describe in the next section. In Section 4.2, we describe the penalty branch-and-bound method that we use to solve that reformulation. In Section 4.3, we present possible problem-specific enhancements of the algorithm, which we test numerically in Section 4.4.

4.1. Another Reformulation for Mixed-Integer Linear Complementarity Problems

As the big-M constants in the reformulation by Gabriel et al. (2013a) are not always obtainable and the complementarity penalty term from Reformulation 2.1.6 is only tractable for monotone MILCP, we come up with another reformulation.

Reformulation 4.1.1 (Non-Convex Penalty Reformulation of a Non-Monotone MILCP)**.** The LCP(q, M, I) can be reformulated with $\alpha \in (0, 1)$ as

$$\min_{z} \quad \alpha \sum_{i=1}^{n} \min\{z_i, (q + Mz)_i\} + (1 - \alpha) \sum_{i \in I} \min\{z_i, 1 - z_i\} \tag{4.1.1a}$$

$$\text{s.t.} \quad z \geq 0, \quad q + Mz \geq 0, \tag{4.1.1b}$$

$$z_I \in [0, 1]^I. \tag{4.1.1c}$$

In a sense, this reformulation is a mixture of the MILP formulation by Gabriel et al. (2013a) and the reformulation we used in Section 2, Reformulation 2.1.6. Again, the parameter α controls the emphasis that is put on each of the two penalty terms. We retain all the properties that we had for the Reformulation 2.1.6, i.e., if and only if the Reformulation 2.1.6 has an optimal solution with objective value of 0, the corresponding non-monotone MILCP has a solution. If the MILCP has no solution, the reformulation has an optimal point that violates the integrality and complementarity constraints as little as possible and is therefore as close as possible to being a solution of the original MILCP.

In the following, we will denote an optimal solution of the problem by z^*, its objective function by f, i.e.,

$$f(z) := \alpha \sum_{i=1}^{n} \min\{z_i, (q + Mz)_i\} + (1 - \alpha) \sum_{i \in I} \min\{z_i, 1 - z_i\},$$

and its feasible set by Z, i.e.,

$$Z := \{z \in \mathbb{R}^n : z \geq 0,\ q + Mz \geq 0,\ z_i \leq 1 \text{ for } i \in I\}.$$

4.2. The Penalty Branch-and-Bound Method for this Reformulation

Instead of only branching on the penalty term for the integrality constraints as we did in Section 2.2, we also have to branch on the penalty term for the complementarity constraints. This results in a greatly increased depth of the branch-and-bound tree. This is not surprising considering the additional non-convexities added by the complementarity constraints of the non-monotone MILCP. As an additional difficulty, the global lower bound at the beginning of the process is zero, as there is no function that is common for every node. Additional to the question, which index we branch on first, we also have to decide whether we want to branch on the integrality or on the complementarity constraints first. As the version of the algorithm is a particular case of Algorithm 3, we will go into less details and only describe the special branching process and afterwards the algorithmic description.

4.2.1. Branching

As there is no term in the objective function of Problem 4.1.1 that is common for all nodes, the root node of the problem is a feasibility problem, as its objective function is 0:

Definition 4.2.1 (Root Node Problem of PBB for Non-Monotone MILCPs)**.** The root node problem of the PBB for the non-monotone LCP(q, M, I) is defined as

$$\min_{z\in\mathbb{R}^n} \quad 0 \tag{4.2.1a}$$
$$\text{s.t.} \quad z \in Z. \tag{4.2.1b}$$

We then start to create child nodes by adding penalty terms for the different types of penalizations. In contrast to the process in Chapter 2, we have two different types of branching decisions and with every branching decision we take, we first have to decide on the branching type. After solving the root node problem, we can either choose binary branching and an index $j \in I$ of a fractional variable and on which we have not yet branched on in the binary sense as we did in Section 2.2, or we choose complementarity branching and an index $j \in [n]$ for which the complementarity constraint is violated and on which we have not yet branched on in the complementarity sense. The process for binary branching is done in the same way as we described before by either adding the penalty term $(1-\alpha)z_j$ or the term $(1-\alpha)(1-z_j)$. The process for complementarity branching is similar. Here we either add the penalty term αz_j or the term $\alpha(q+Mz)_j$. Therefore, there are the following possibilities as child nodes for the root node. If we have chosen complementarity branching, the first new node problems are the following:

Definition 4.2.2 (Left Complementarity Child Problem of the Root Node of PBB for Problem 4.1.1)**.** The left complementarity child problem of the root node of PBB for the non-monotone LCP(q, M, I) is defined as

$$\min_{z\in\mathbb{R}^n} \quad \alpha z_j \tag{4.2.2a}$$
$$\text{s.t.} \quad z \in Z, \tag{4.2.2b}$$

Definition 4.2.3 (Right Complementarity Child Problem of the Root Node of PBB for Problem 4.1.1)**.** The right complementarity child problem of the root node of PBB for the non-monotone LCP(q, M, I) is defined as

$$\min_{z\in\mathbb{R}^n} \quad \alpha(q+Mz)_j \tag{4.2.3a}$$
$$\text{s.t.} \quad z \in Z. \tag{4.2.3b}$$

For binary branching the first new node problems are the following two.

Definition 4.2.4 (Left Binary Child Problem of the Root Node of PBB for Problem 4.1.1)**.** The left binary child problem of the root node of PBB for the non-monotone LCP(q, M, I) is defined as

$$\min_{z\in\mathbb{R}^n} \quad (1-\alpha)z_j \tag{4.2.4a}$$
$$\text{s.t.} \quad z \in Z, \tag{4.2.4b}$$

Definition 4.2.5 (Right Binary Child Problem of the Root Node of PBB for Problem 4.1.1)**.** The right binary child problem of the root node of PBB for the non-monotone LCP(q, M, I) is defined as

$$\min_{z \in \mathbb{R}^n} \quad (1-\alpha)(1-z_j) \tag{4.2.5a}$$
$$\text{s.t.} \quad z \in Z. \tag{4.2.5b}$$

As described in the previous chapters, these new problems are solved independently and more penalty terms are added successively, until there are no more branching candidates.

In the following we will refer to the process of creating a left binary child node as "downwards" branching, the process of creating a right binary child node as "upwards" branching, the process of creating a left complementarity child node as "leftwards" branching and the process of creating a right complementarity child node as "rightwards" branching.

For binary branching, we will again denote the indices on which we have already branched on downwards as I_0 and the indices we have already branched on upwards as I_1. For the complementarity branching, we denote the indices we have already branched on leftwards by C_0 and the indices for which we have already branched on rightwards as C_1. With this notation, we can define any node in the branch-and-bound tree by the tuple (I_0, I_1, C_0, C_1) and the node problem of a node $N = (I_0, I_1, C_0, C_1)$ is the following:

Definition 4.2.6 (Node Problem of PBB for Problem 4.1.1)**.** The node problem of the PBB for the non-monotone LCP(q, M, I) at node $N = (I_0, I_1, C_0, C_1)$ is defined as

$$\min_{z \in \mathbb{R}^n} \quad \alpha \sum_{i \in C_0} z_i + \alpha \sum_{i \in C_1} (q + Mz)_i + (1-\alpha) \sum_{i \in I_0} z_i + (1-\alpha) \sum_{i \in I_1} (1 - z_i) \tag{4.2.6a}$$
$$\text{s.t.} \quad z \in Z. \tag{4.2.6b}$$

In the following, we will refer to the objective function of the node problem at node $N = (I_0, I_1, C_0, C_1)$ as f_N, i.e.,

$$f_N(z) := \alpha \sum_{i \in C_0} z_i + \alpha \sum_{i \in C_1} (q + Mz)_i + (1-\alpha) \sum_{i \in I_0} z_i + (1-\alpha) \sum_{i \in I_1} (1 - z_i).$$

4.2.2. Algorithmic Description

We will now formally state the scheme of the penalty branch-and-bound method for non-monotone MILCP in Algorithm 5.

As we have already proven the finiteness and correctness of a more general version of the algorithm, we can prove correctness and finiteness for this version by reduction.

Theorem 4.2.7. *Algorithm 5 terminates after finitely many steps with a global optimal solution of Problem 4.1.1.*

Proof. Let $J = ([n], m) = \{1^{m(1)}, \dots, n^{m(n)}\}$ be the multiset of all indices with $m(i) = 1$ for all $i \notin I$ and $m(i) = 2$ for all $i \in I$. Let further $J_i = \{1, 2\}$ for all $i \in J$ and $g_{i1}(z) = \alpha z_i$ for

Algorithm 5 A Penalty Branch-and-Bound Algorithm for Non-Monotone MILCPs

Input: $q \in \mathbb{R}^n$, $M \in \mathbb{R}^{n\times n}$, $I \subseteq [n]$, $\alpha \in (0,1)$
Output: A global optimum z^* of Problem 4.1.1.
Set $\mathcal{N} \leftarrow \{(\emptyset,\emptyset,\emptyset,\emptyset)\}$, $f_{\text{inc}} \leftarrow \infty$, and $z^*_{\text{inc}} \leftarrow \mathsf{none}$.
while $\mathcal{N} \neq \emptyset$ **do**
  Choose $N = (I_0, I_1, C_0, C_1) \in \mathcal{N}$ and set $\mathcal{N} \leftarrow \mathcal{N} \setminus \{N\}$.
  Compute $z^*_N \in \operatorname{argmin}\{f_N(z) : z \in Z\}$.
  if $f(z^*_N) < f_{\text{inc}}$ **do**
    Set $z^*_{\text{inc}} \leftarrow z^*_N$ and $f_{\text{inc}} \leftarrow f(z^*_N)$.
  if $f_N(z^*_N) < f_{\text{inc}}$ **and** $(I \setminus (I_0 \cup I_1) \neq \emptyset$ **or** $([n] \setminus (C_0 \cup C_1) \neq \emptyset)$ **do**
    Choose $j_1 \in I \setminus (I_0 \cup I_1)$ and
    set $\mathcal{N} \leftarrow \mathcal{N} \cup \{(I_0 \cup \{j_1\}, I_1, C_0, C_1), (I_0, I_1 \cup \{j_1\}, C_0, C_1)\}$
    or
    Choose $j_2 \in [n] \setminus (C_0 \cup C_1)$ and
    set $\mathcal{N} \leftarrow \mathcal{N} \cup \{(I_0, I_1, C_0 \cup \{j_2\}, C_1), (I_0, I_1, C_0, C_1 \cup \{j_2\})\}$.
return z^*_{inc}

every first occurrence of the index i in the multiset J and $g_{i1}(z) = (1-\alpha)z_i$ for every second occurrence. Let $g_{i2}(z) = \alpha(q + Mz)_i$ for every first occurrence of the index i in the multiset J and $g_{i2}(z) = (1-\alpha)(1-z_i)$ for every second occurrence. Let $h(z) = 0$ and $X = Z$. Now, Problem 4.1.1 is in the form of Problem 3.1.1 and correctness of Algorithm 5 follows from Theorem 3.1.8. □

4.3. Further Enhancements of the Method

Again, there are different ways to improve the performance of our algorithm. We have not specified how to choose the next branching index $j_1 \in I$ or $j_2 \in [n]$ or how to choose the next unsolved subproblem N in Algorithm 5. Again, there are also different ways to compute the optimal solution of a node problem. For example, we can warmstart each node to solve the node quicker or we can implement cutting planes to improve lower bounds.

4.3.1. Choosing the Branching Index

In most branch-and-bound methods, the branching is solely done on the binary or integrality constraints. Here, we have a different situation. In this special case, we not only need to decide on an index, but we also need to decide on the type of branching.

For the choice of the branching type, we propose four different strategies. We can either choose the type randomly, we can choose complementarity branching for as long as there are candidates and only then consider binary branching, we can do the same with reversed roles or we can base the choice of branching type on the same score that we use to choose the index to branch on as we did with pseudocost and MIQP-based branching in Section 2.3.

For the choice of the branching index we propose two branching strategies, that we have already used for the monotonic case and that are well known from classic branch-and-bound methods. The first is a direct adaption of “pseudocost branching”. Unfortunately, there is no direct analogy for our “MIQP-based branching” from Section 2.3.1, which is why we have to modify that approach. We call that approach “preprocessed order branching”. We will also consider the random choice of the index and choosing the constraint that is violated the most as benchmark strategies.

In our numerical experiments, we compare different combinations of type and index choice strategies. For the benchmark strategies we use the following combinations. For the random index choice strategy, we also choose the type of branching at random with a proportional probability, i.e., the probability of choosing binary branching is the ratio of the number of binary branching candidates to the number of complementarity branching candidates. For the most-violated index choice, we test two different approaches, one where we start by branching on all binary candidates first and then on the complementarity branching candidates and one where we do it the other way around. For the score-based strategies that we will discuss in the following, where we use the candidate with the highest score, we will also use the corresponding branching type, i.e., we compute the scores for all candidates of both branching types and choose the index and type of the highest scoring constraint.

Pseudocost Branching

We have already described the principles of pseudo-cost branching in Section 2.3.1 and we will describe the adjustments for this specific problem now.

Let $\varphi^1_{N,j}$ be the objective gain per unit change when we branch upwards on variable $j \in I$ at node N:

$$\varphi^1_{N,j} := \frac{f(z^*_{N_1}) - f(z^*_N)}{\lceil (z^*_{N_1})_j \rceil - (z^*_{N_1})_j}.$$

Here, N_1 is the child of N created by upwards branching. We denote by ψ^1_j the expected objective gain per unit change when we branch 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 ψ^1_j as

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

Analogously, we can define $\varphi^0_{N,j}$ and ψ^0_j for downwards branching on variable $j \in I$, $\varphi^l_{N,j}$ and ψ^l_j for leftward branching on variable $j \in [n]$ and $\varphi^r_{N,j}$ and ψ^r_j for rightward branching on variable $j \in [n]$, but have to change the denominator for the definition of the different $\varphi_{N,j}$. For downwards branching, the denominator has to be $(z^*_{N_0})_j - \lfloor (z^*_{N_0})_j \rfloor$, for leftwards branching the denominator is just $(z^*_{N_l})_j$ and for rightwards branching it is $(q + M(z^*_{N_r})_j)_i$.

The average gain is then calculated as

$$s_j^I := \mu \min\left\{\psi_j^0 \cdot ((z^*_{N_0})_j - \lfloor (z^*_{N_0})_j \rfloor),\, \psi_j^1 \cdot (\lceil (z^*_{N_1})_j \rceil - (z^*_{N_1})_j)\right\}$$
$$+ (1-\mu) \max\left\{\psi_j^0 \cdot ((z^*_{N_0})_j - \lfloor (z^*_{N_0})_j \rfloor),\, \psi_j^1 \cdot (\lceil (z^*_{N_1})_j \rceil - (z^*_{N_1})_j)\right\}$$

with $\mu \in (0,1)$ for binary branching and

$$s_j^C := \mu \min\left\{\psi_j^l \cdot (z^*_{N_l})_j,\, \psi_j^r \cdot (q + M(z^*_{N_r})_j)_i\right\}$$
$$+ (1-\mu) \max\left\{\psi_j^l \cdot (z^*_{N_l})_j,\, \psi_j^r \cdot (q + M(z^*_{N_r})_j)_i\right\}$$

with $\mu \in (0,1)$ for complementarity branching. Then, the pseudocost-based branching candidate is the branching type and index j with the largest score s_j^I or s_j^C. At the beginning of our branch-and-bound, we initialize the average $\psi_j^{0,1,l,r}$ with 1. If at a certain node N, we have not yet branched on a candidate j, namely $N^j = \emptyset$, we initialize $\psi_j^{0,1,l,r}$ with the average of all other $\psi_i^{0,1,l,r}$ for $i \in I$ with $i \neq j$ or $i \in [n]$ with $i \neq j$ respectively for which $N^i \neq \emptyset$.

Preprocessed Order Branching

We want to adapt the MIQP-based approach from Section 2.3.1 as the resulting MIQPs used would not be convex for non-monotone MILCP. Again, we propose a strategy based on solving an optimization problem for each branching candidate in the presolve phase of the algorithm. Again, we aim at sorting the indices of variables so that we branch on those indices first that are expected to give good lower bounds on the optimal solution. For every index $j \in [n]$, we solve the following problem with a single integer variable:

$$\min_{z \in \mathbb{R}^n} \quad \min\{z_j, (q+Mz)_j\} \tag{4.3.1a}$$
$$\text{s.t.} \quad q + Mz \geq 0,\ z \geq 0, \tag{4.3.1b}$$
$$z_j \in \{0,1\}. \tag{4.3.1c}$$

We achieve this, by solving two MILPs, one in which the objective function is z_i and one in which the objective function is $(q+Mz)_i$, and taking the minimal objective value of both optimal solutions. For indices $i \notin I$ Constraint (4.3.1c) is omitted. As discussed before, we know that it is likely that the overall MILCP has no solution and that this is due to the combination of complementarity and integrality constraints. By solving all $2n$ many MILPs (4.3.1) we measure the impact of the every variable on the infeasibility of the problem (if it is infeasible at all). The indices $j \in [n]$ are then sorted with decreasing optimal objective function values of Problem (4.3.1). Moreover, infeasible problems are formally assigned the objective function value ∞. 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 and performs both binary and complementarity branching (if possible).

4.3.2. Choosing the Next Subproblem to Solve

In our implementation of the penalty branch-and-bound algorithm, we consider the same three different node selection strategies as before. The first two are depth- and breadth-first search. The third strategy will again be referred to as the "lower bound push strategy".

From Lemma 3.1.7, 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 take the left child if $z_i \leq 0.5$ and integer branching was chosen with i being the last branching index or if $z_i \leq (q + Mz)_i$ and complementarity branching was chosen with i being the last branching index and the right child otherwise. 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.

4.3.3. Warmstarting the Node Problems

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 warmstarting the LP 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.

4.3.4. Implementing Valid Inequalities

From Theorem 3.1.11 we know that we can add simple cuts without changing the correctness of the algorithm. In our case, there are two different types of simple cuts. For every index $j \in I$ that we have branched on binary and downwards in a node N, we can include $z_j \leq 0.5$ as a constraint in that node. This is because any point violating that cut would yield an even better result in the sibling node that is the same only with upwards branching. The same holds analogously for upwards branching on index $j \in I$ and the cut $z_j \geq 0.5$. For the complementarity branching, the cuts are $z_j \leq (q + Mz)_j$ for any index $j \in [n]$ that we have branched on leftwards and $z_j \geq (q + Mz)_j$ for any index $j \in [n]$ that we have branched on rightwards. Correctness of these cuts follows from Theorem 3.1.11. We will test the inclusion of different combinations of these cuts.

4.4. Numerical Results

In Section 4.3 we proposed various ways to improve the overall performance of our method. In this section, we will present the results of the numerical experiments we conducted to compare the different techniques performance-wise. We tested all types of enhancements independently of each other. For every test we take, the best settings from previous tests together with "standard" settings for untested parameters is used. We start with a test of the different branching rules in Section 4.4.1, followed by a test of the node selection strategies in Section 4.4.2, different warm starting techniques in Section 4.4.3, and different strategies for the inclusion of valid inequalities in Section 4.4.4. Afterwards, we test our method with the best setting we identified against a benchmark approach.

The test setup is the same as for the numerical experiments in Section 2.4. We implemented the penalty branch-and-bound method presented in Section 4.2 in Python 3.7. All node problems are solved with the LP solver of Gurobi 9.1.2 and all the tests were run on an Intel Xeon CPU E5-2699 v4 @ 2.20 GHz (88 cores) with 756 GB RAM. In this section, MILCP-PBB refers to the implementation of Algorithm 5. The test instances we use are constructed at randomly in a similar fashion to the instances of Section 2.4. The matrices $M \in \mathbb{R}^{n \times n}$ have been created using the sprandsym function of MATLAB for sizes

$$n \in \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}.$$

We then again 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 2.1.3. Then, it satisfies

(i) Feasibility w.r.t. Z: $z^* \in Z$,

(ii) Integrality: $z_i^* \in \{0, 1\}$ for all $i \in I$,

(iii) Complementarity: $(z^*)^\top (q + Mz^*) = 0$.

The vectors q have been created to satisfy at least one of the conditions above. Again, 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)–(d), yielding 400 different instances in total. More details on how the test set has been built can be found in Appendix D.

For the comparisons presented in this section we again use logarithmic performance profiles in the sense of Dolan and Moré (2002) as well as tables with the most important statistical measures. For the tables, we aggregated all instances that have been solved by all parameter settings or solution approaches for the specific test w.r.t. the instance size. The first column always states the dimension n of the problem. The second column contains the arithmetic mean of node counts and running times respectively for all instances solved by every parameterization. The next columns contain the median, the minimum, and the maximum value of the data set. The sixth and seventh column contain the 0.25-quantile, i.e., the node count or running time after which 25 % of instances were solved, as well as the 0.75-quantile. The next column contains the geometric shifted mean. The shift is 100 for the node counts and 10 for the running times. The last column contains the percentage of instances solved to global optimality for the parameterization and instance size. The best value for every measure and instance size among all tables for that test is printed bold. The table of the winning setting, i.e., the best performing parameterization, is included in this section whereas the tables of the other settings are included in Appendix E.

The timelimit for these tests is set to 1 h.

4.4.1. The Impact of Different Branching Rules

We now compare the performance of MILCP-PBB when equipped with the five different branching rules. Two of these strategies are the pseudocost branching rule and the preprocessed order branching described in Section 4.3.1, where we also choose the branching type according to the score. The other three are the naive approaches described in Section 4.3.1. One is a random branching decision, where the decision between complementarity and integer branching is made at random but proportional to the number of possible branching candidates. The other two approaches are strategies, where the index of the most-violated constraint is used as the branching index. One time we start with all possible complementarity branching decisions and then all integer branching decisions and the second time we do it the other way around. 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 83 instances from the test set since no parameterization was able to solve them within the time limit. Figure 4.1 displays the performance profiles w.r.t. the required number of branch-and-bound nodes (left figure) and running times (right figure).

One can see that the performance of the preprocessing order branching is the worst both in regards to the nodecount and the runtime, with the random choice being only slightly better. The pseudocost branching and the most violated approach, where integer branching is done first, are very close performance-wise, while the most violated approach, where complementarity branching is done first, is the clear winner.

The conclusions that can be drawn from the statistical measures as displayed in Table 4.1 (and Tables E.1–E.4 in the appendix) are less unambiguous with a wider spread of best results among methods. It also has to be noted, that the sample size for the larger instance sizes are rather small, for example for $n = 60$ only one instance was solved by all methods.

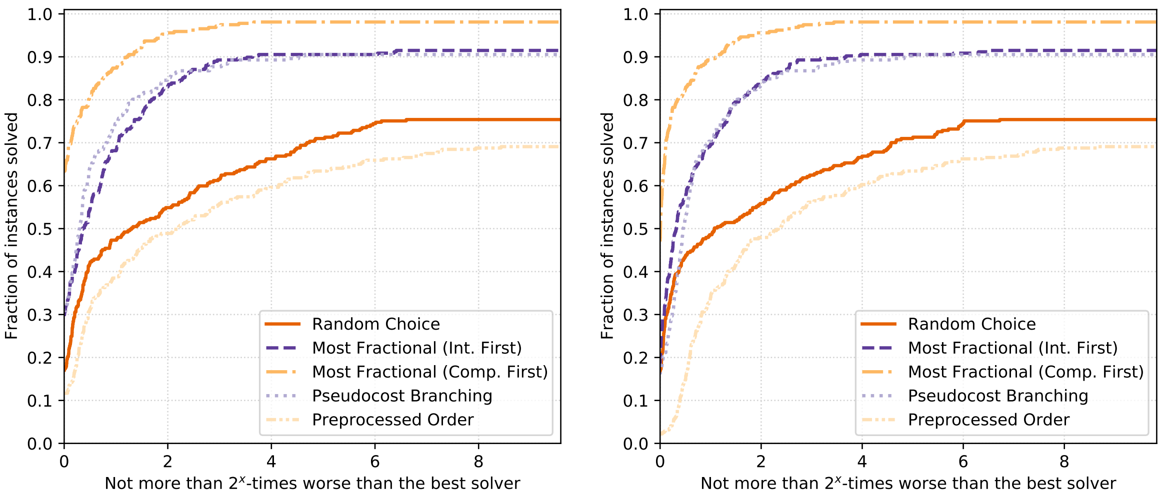


Figure 4.1.: Performance profiles for the number of branch-and-bound nodes (left) and the running time (right) of all branching rules

Nevertheless, the most-violated branching rule with complementarity branching done first appears to be the best and for some measures it can be seen, that both random branching and our preprocessing rule are significantly worse.

Table 4.1.: Aggregated nodecounts (top) and runtimes (bottom) for the branching rule test with most fractional branching and complementarity branching done first

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
10	6.0	**5.0**	**1.0**	31.0	**3.0**	**7.0**	5.9	**100**
20	49.2	**18.0**	**5.0**	345.0	**11.0**	56.5	38.6	**100**
30	431.1	**149.0**	33.0	3545.0	**49.0**	**429.0**	227.4	**100**
40	1726.7	**207.0**	**33.0**	10255.0	**64.0**	651.0	420.0	**100**
50	**141.0**	**75.0**	**65.0**	**517.0**	**75.0**	**90.0**	**112.7**	**100**
60	**87.0**	**87.0**	**87.0**	**87.0**	**87.0**	**87.0**	**87.0**	**85**
10	**0.0**	**0.0**	**0.0**	0.1	**0.0**	**0.0**	**0.0**	**100**
20	**0.2**	**0.1**	**0.0**	1.4	**0.1**	**0.2**	**0.2**	**100**
30	3.9	**1.4**	0.3	32.8	**0.5**	**3.8**	3.0	**100**
40	31.0	**3.6**	**0.7**	188.4	1.3	**11.4**	11.7	**100**
50	**4.1**	**2.5**	**2.3**	**13.8**	**2.4**	**2.7**	**3.7**	**100**
60	**4.4**	**4.4**	**4.4**	**4.4**	**4.4**	**4.4**	**4.4**	**85**

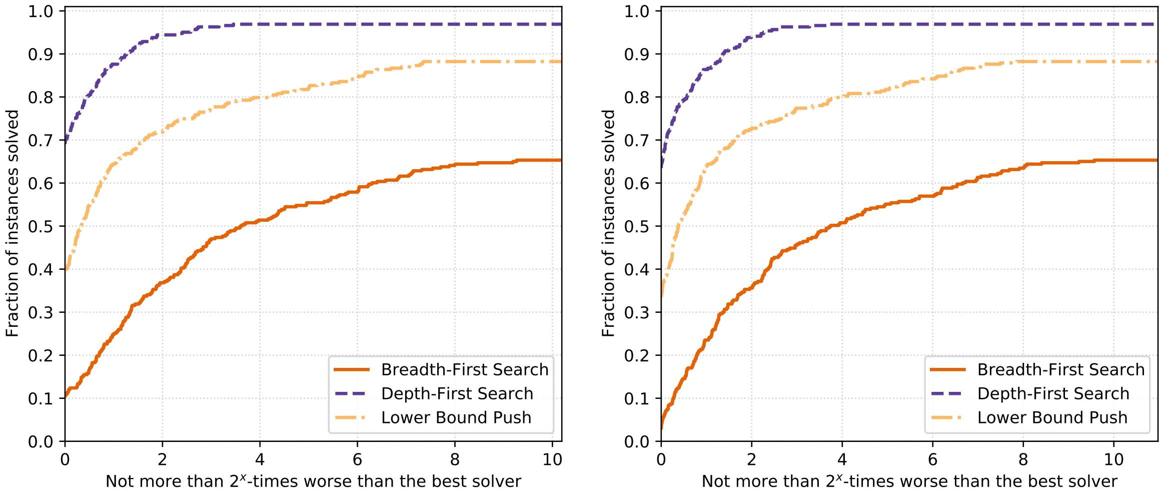


Figure 4.2.: Performance profiles on the number of branch-and-bound nodes (left) and the running time (right) of all node selection strategies.

4.4.2. The Impact of Different Node Selection Strategies

We now compare the three node selection strategies described in Section 4.3.2. To this end, we use the most violated branching strategy with complementarity branching done first, while warmstarts and valid inequalities are disabled. We exclude 77 instances from the set since no parameterization of our method is able to solve them within the time limit. It can be seen in Figure 4.2, that for this test the choice of the node selection strategy had a significant impact on the performance, with the depth-first search approach being the clear winner and our lower bound push strategy being the clear runner-up.

The statistical measures we present in Tables 4.2, E.5, and E.6 support the conclusions. For most measures the depth-first search strategy performs best, followed by the lower-bound-push strategy.

4.4.3. The Impact of Warmstarts

We now compare the performance of MILCP-PBB with and without warmstarts. To this end, we use the most violated branching strategy with complementarity branching done first, the depth-first search node selection strategy, and avoid the use of any valid inequalities. Again, we tried two different techniques within Gurobi to warm start the node problems. First, we used the Gurobi attributes VBasis and CBasis, i.e., we started every node problem with the optimal basis of its parent node. Second, we used the attributes PStart and DStart, where the optimal basis vector of the parent node is computed from the optimal solution. In case that warmstarts are used, we need to solve the node problems using the primal simplex method within Gurobi. We exclude 75 instances from the set as no parameterization is able to solve them within the time limit. Unfortunately, warmstarts do not help to reduce the running time; see Figure 4.3 (right). Again, the difference in the nodecount comes from

Table 4.2.: Aggregated nodecounts (top) and runtimes (bottom) for the node selection test with depth-first search

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
10	**6.0**	5.0	**1.0**	**31.0**	**3.0**	**7.0**	**5.9**	**100**
20	49.2	**18.0**	**5.0**	345.0	**11.0**	56.5	38.6	**100**
30	431.1	**149.0**	**33.0**	3545.0	**49.0**	429.0	227.4	**100**
40	1726.7	**207.0**	**33.0**	10255.0	**64.0**	**651.0**	**420.0**	**100**
50	**141.0**	**75.0**	**65.0**	**517.0**	**75.0**	**90.0**	**112.7**	**100**
60	**87.0**	**87.0**	**87.0**	**87.0**	**87.0**	**87.0**	**87.0**	85
10	**0.0**	**0.0**	**0.0**	**0.1**	**0.0**	**0.0**	**0.0**	**100**
20	**0.2**	**0.1**	**0.0**	1.3	**0.0**	0.2	**0.2**	**100**
30	3.6	**1.4**	**0.2**	30.5	**0.4**	**3.3**	2.8	**100**
40	27.0	**3.2**	**0.3**	168.2	**1.2**	**8.1**	**10.7**	**100**
50	**3.9**	**2.4**	**1.9**	**13.1**	**2.2**	**2.9**	**3.6**	**100**
60	**3.0**	**3.0**	**3.0**	**3.0**	**3.0**	**3.0**	**3.0**	85

the different solution methods used for the node problems, which might result in different optimal solutions for 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. The picture that Tables 4.3, E.7, and E.8 paint is a little different. Here, parameters VBasis and CBasis are best for most measures and sizes, but as there is no clear improvement overall, we will continue without warmstarts.

Table 4.3.: Aggregated nodecounts (top) and runtimes (bottom) for the warm start test using VBasis/CBasis

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
10	**5.2**	**4.0**	**1.0**	23.0	**3.0**	**5.5**	**5.1**	**100**
20	**27.5**	**15.0**	**3.0**	**165.0**	11.0	**29.5**	**24.4**	**100**
30	286.0	151.0	**7.0**	2193.0	**41.0**	**315.0**	**176.9**	**100**
40	930.0	**169.0**	29.0	**5531.0**	**53.0**	**703.0**	**314.2**	**100**
50	196.7	**65.0**	**53.0**	983.0	**59.0**	**79.0**	116.1	**100**
60	**81.0**	**81.0**	**81.0**	**81.0**	**81.0**	**81.0**	**81.0**	88
10	**0.0**	**0.0**	**0.0**	**0.1**	**0.0**	**0.0**	**0.0**	**100**
20	**0.1**	**0.1**	**0.0**	**0.7**	0.1	**0.1**	**0.1**	**100**
30	2.7	1.5	**0.1**	21.2	**0.4**	**2.9**	**2.3**	**100**
40	17.3	**2.8**	**0.6**	**107.1**	**1.0**	12.2	**8.5**	**100**
50	5.8	**2.1**	**1.7**	28.1	**2.0**	**2.5**	4.3	**100**
60	**4.3**	**4.3**	**4.3**	**4.3**	**4.3**	**4.3**	**4.3**	88

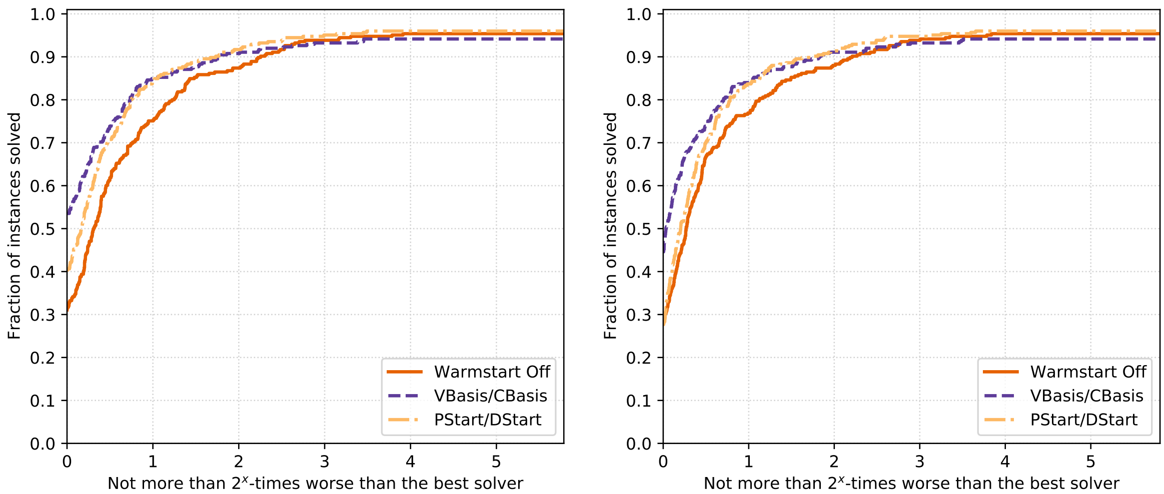


Figure 4.3.: Performance profiles for the number of branch-and-bound nodes (left) and the running time (right) of the warmstart test.

4.4.4. The Impact of the Inclusion of Valid Inequalities

We tested four different settings for the valid inequalities test with the cuts described in Section 4.3.4. We compared our method without cuts, with all binary cuts, all complementarity cuts, and all simple cuts. The binary cuts can be set up at no computational cost and therefore 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. The inclusion of the complementarity cuts might lead to higher computational costs, as they are dense linear inequalities in general. For this test, the branching rule is set to the most violated branching strategy with complementarity branching done first, the node selection strategy is set to depth-first search, and warmstarts are disabled. 88 instances were excluded for this test. As it can be seen in Figure 4.4, incorporating the binary cuts has almost no impact both on the number of branch-and-bound nodes as well as on the running time, while the complementarity cuts slow down the solution process significantly. From Tables 4.4 and E.9 to E.11 we can see, that this is not due to the number of nodes needed, as the measures are rather similar, but that the runtime increases so much that a lot of the more difficult instances cannot be solved within the timelimit. In the tables, the version without any cuts dominates slightly, which is why we choose that setting for the following test.

4.4.5. Testing the Method Against a Commercial Benchmark Approach

From the preliminary numerical tests we know that the best parameterization of MILCP-PBB uses the most violated branching strategy with complementarity branching done first, uses depth-first-search as the node selection strategy, adds no simple cuts and warmstarts are disabled.

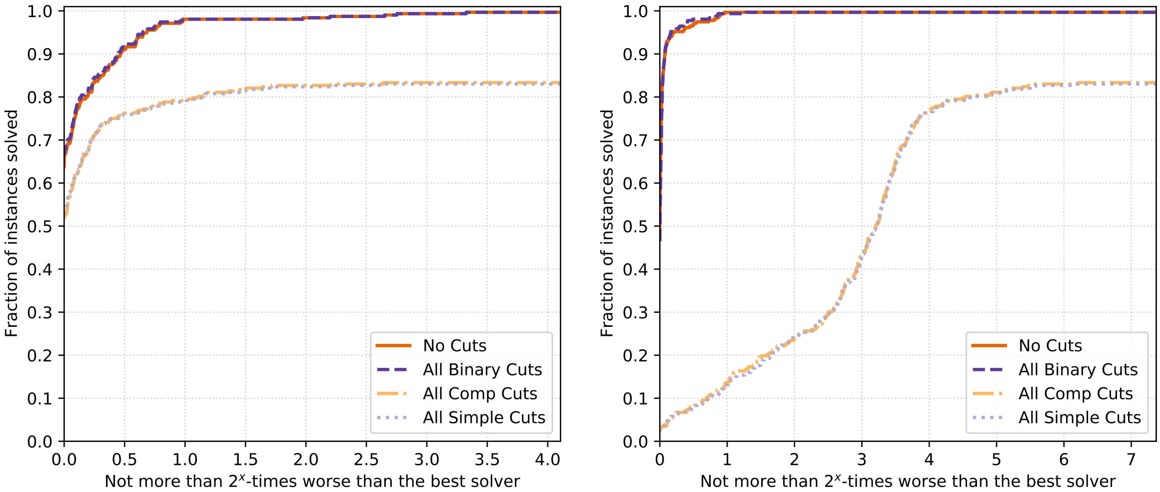


Figure 4.4.: Performance profiles for the number of branch-and-bounds nodes (left) and the running time (right) for the valid inequality test.

Table 4.4.: Aggregated nodecounts (top) and runtimes (bottom) for the valid inequality test without cuts

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
10	6.0	**5.0**	**1.0**	31.0	**3.0**	**7.0**	5.9	**100**
20	49.2	**18.0**	**5.0**	345.0	**11.0**	**56.5**	**38.6**	**100**
30	431.1	**149.0**	**33.0**	3545.0	**49.0**	**429.0**	**227.4**	**100**
40	1726.7	**207.0**	33.0	**10255.0**	**64.0**	651.0	420.0	**100**
50	141.0	**75.0**	65.0	517.0	**75.0**	**90.0**	112.7	**100**
60	87.0	87.0	87.0	87.0	87.0	87.0	87.0	**85**
10	**0.0**	**0.0**	**0.0**	**0.1**	**0.0**	**0.1**	**0.0**	**100**
20	**0.2**	**0.1**	**0.0**	**1.4**	**0.1**	**0.2**	**0.2**	**100**
30	**4.0**	**1.5**	**0.3**	**33.1**	**0.5**	**4.0**	**3.1**	**100**
40	**31.9**	**3.6**	**0.6**	**190.3**	**1.3**	11.7	12.0	**100**
50	**4.1**	**2.4**	2.2	13.7	**2.4**	**2.9**	**3.7**	**100**
60	**4.6**	**4.6**	**4.6**	**4.6**	**4.6**	**4.6**	**4.6**	**85**

In order to compare MILCP-PBB with other approaches from the literature, we again consider the MILP Reformulation 2.1.5, that was proposed in Gabriel et al. (2013a). Remember, that the drawback of Model 2.1.5 is that it requires to determine sufficiently large big-B constraints. For the test we used $B = 10^5$. When 2.1.5 is solved using Gurobi, all presolve techniques and heuristics have been disabled. For obtaining a fair comparison, we further restrict both the MILP solver of Gurobi and the LP solver of Gurobi used for solving the nodes within MILCP-PBB to only use a single thread. For this test, 88 instances are excluded.

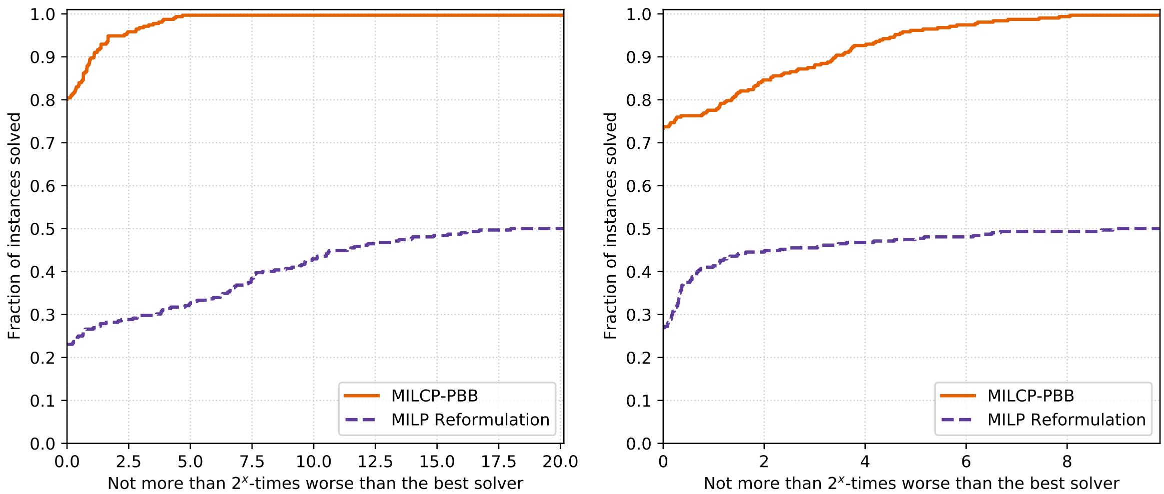


Figure 4.5.: Performance profiles for the number of branch-and-bound nodes (left) and the running time (right) for the MILP reformulation and MILCP-PBB.

Table 4.5.: Aggregated nodecounts (top) and runtimes (bottom) for the solver test for MILCP-PBB

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
10	**6.0**	**5.0**	**1.0**	**31.0**	3.0	7.0	**5.9**	**100**
20	**49.2**	**18.0**	5.0	**345.0**	11.0	56.5	**38.6**	**100**
30	**431.1**	**149.0**	33.0	**3545.0**	49.0	**429.0**	**227.4**	**100**
40	**1726.7**	**207.0**	33.0	**10255.0**	**64.0**	**651.0**	**420.0**	**100**
50	**141.0**	**75.0**	**65.0**	**517.0**	**75.0**	**90.0**	**112.7**	**100**
60	**87.0**	**87.0**	**87.0**	**87.0**	**87.0**	**87.0**	**87.0**	**85**
10	**0.1**	**0.1**	**0.0**	**0.1**	**0.1**	**0.1**	**0.1**	**100**
20	0.2	0.1	**0.0**	1.4	0.1	0.2	0.2	**100**
30	4.0	1.4	0.3	33.5	0.4	3.9	3.1	**100**
40	**32.0**	**3.5**	0.6	**193.1**	1.2	**11.1**	11.9	**100**
50	**4.2**	**2.7**	2.2	**14.0**	**2.4**	**2.8**	**3.8**	**100**
60	**4.7**	**4.7**	**4.7**	**4.7**	**4.7**	**4.7**	**4.7**	**85**

Figure 4.5 shows the performance profiles of MILCP-PBB and Gurobi for the MILP w.r.t. the number of nodes and running times. It can be seen that MILCP-PBB needs both significantly fewer nodes, and less runtime. More details can be found in Tables 4.5 and E.12. It is evident that our approach clearly outperforms the benchmark approaches for all measures and sizes, even though the MILP reformulation is faster on the easier instances, probably due to inefficiencies in our Python implementation.

Chapter 5

Tackling Mixed-Integer Linear Problems

As the class of MILCP is a rather specific one, we want to investigate a more general class of problems in detail. As mentioned before, one of the biggest and most important problem classes is the class of MILP. We have already described these problems in Definition 1.1.1. MILPs are probably the most and best studied class of optimization problems, so improvements would be a huge achievement. For MILPs, there are a lot of different solution approaches and problem specific enhancements for branch-and-bound methods, such as a zoo of different valid inequalities, branching rules, node selection strategies and heuristics. We want to add another possible solution method to this research and will therefore study different aspects of solving these problems with our penalty branch-and-bound methods. In Section 5.1 we give a penalty reformulation of MILP that we can solve with our method and we will present different theoretical results on the possibilities of finding exact solutions as well as possibilities to establish lower and upper bounds on the solution of MILP. In Section 5.2 we present different ways on how we might be able to improve the performance of our method and in Section 5.3 we present the results of different numerical experiments. In that section, we test the different possible enhancements and compare our method to different benchmark approaches for computing lower bounds, upper bounds and exact solutions both in the sense of runtime and quality of the bounds found. In this section, we assume that the MILPs and their linear relaxations are bounden from below.

5.1. Reformulating Mixed-Integer Linear Problem with Penalty Terms

We have already discussed the general principles of penalty reformulations in Section 1.1.2. In order to use our penalty branch-and-bound method, we can again replace the binary constraints by the penalty function P_I introduced in Section 1.1.2 as we already did in Chapters 2 and 4. This leads to the following reformulation.

Reformulation 5.1.1 (Penalty Reformulation for MILPs)**.** The MILP 1.1.1 can be reformulated as

$$\min_{x \in \mathbb{R}^n} \quad c^\top x + \mu \sum_{i \in I} \min\{x_i, 1 - x_i\} \tag{5.1.1a}$$

$$\text{s.t.} \quad Ax \le b, \tag{5.1.1b}$$

$$x \ge 0, \tag{5.1.1c}$$

$$x_I \in [0,1]^I, \tag{5.1.1d}$$

with $\mu \in \mathbb{R}_{>0}$ being sufficiently large.

One can see that this reformulation is very similar to Reformulation 2.1.6. Only the first term of the objective function is linear instead of quadratic and the feasible set is a different polytope. Therefore, we can apply a lot of the ideas from Chapter 2 to this problem class.

As we mentioned before, the value of μ is relevant for the exactness of the reformulation. We first want to show, that independent of the value of μ, Reformulation 5.1.1 delivers a lower bound on the optimal objective value of Problem 1.1.1.

Lemma 5.1.2. *The optimal objective value of Reformulation 5.1.1 is a lower bound for the optimal objective value of Problem 1.1.1 for any $\mu > 0$.*

Proof. Let x^* be an optimal solution of Problem 1.1.1 and $\bar{x}$ an optimal solution of Reformulation 5.1.1. Let us assume that our claim is not true and that

$$c^\top \bar{x} + \mu P_I(\bar{x}) > c^\top x^*.$$

Because x^* is integer feasible, we have

$$c^\top \bar{x} + \mu P_I(\bar{x}) > c^\top x^* + \mu P_I(x^*),$$

which is a contradiction to the optimality of $\bar{x}$. □

The next big question big question is the existence of a large-enough μ and how to identify such a μ. For binary-constrained optimization problems with a linear objective function and a convex feasible set, sufficient conditions have been presented for example in De Santis and Rinaldi (2012). As MILPs are of that form, we can use these conditions and simplify them.

In what follows, we will use $P := \{x \in \mathbb{R}^n_{\ge 0} \colon Ax \le b,\ x_I \in [0,1]^I\}$ as the feasible set of the linear relaxation of a MILP as described in Definition 1.1.2.

Lemma 5.1.3. *Let P be a polytope with at least one fractional extreme point. Let u be an upper bound on the optimal value of Problem 1.1.1 and l be the optimal objective value of the linear relaxation. Let $\varepsilon > 0$ be such that $\underline{x} > \varepsilon$ and $1 - \bar{x} > \varepsilon$ with*

$$\underline{x} := \min_{x \in S} L(x), \quad \bar{x} := \max_{x \in S} U(x),$$

where

$$L(x) := \begin{cases} \min\{x_i \colon i \in I,\ x_i \neq 0\}, \text{ if } x_I \neq 0, \\ 1, \text{ else}, \end{cases}$$
$$U(x) := \begin{cases} \max\{x_i \colon i \in I,\ x_i \neq 1\}, \text{ if } x_I \neq 1, \\ 0, \text{ else}. \end{cases}$$

and $S \subset \mathbb{R}^n$ is the set of extreme points of P. Then for $\mu > \frac{u-l}{\varepsilon}$ any optimal solution of Reformulation 5.1.1 constitutes an upper bound on the optimal solution of the MILP 1.1.1, if the MILP is feasible.

Proof. Let $\mu \in \mathbb{R}$ be such that $\mu > \frac{u-l}{\varepsilon}$. Let $x' \in S$ be such that there exists a $j \in I$ with $x'_j \notin \{0,1\}$. We then have

$$\begin{aligned} P_I(x') &\geq \min\{x'_j, 1 - x'_j\} \\ &\geq \min\{L(x'), 1 - U(x')\} \\ &\geq \min\{\min_{x \in S} L(x), 1 - \max_{x \in S} U(x)\} \\ &= \min\{\underline{x}, 1 - \bar{x}\} > \varepsilon. \end{aligned}$$

Hence we have that
$$\frac{P_I(x)}{\varepsilon} > 1 \text{ for all } x \in S \text{ with } x_I \notin \{0,1\}^I.$$

Now let x^* be an optimal solution of Problem 1.1.1. We first assume, that there is a point $\bar{x} \in S$, for which
$$c^\top \bar{x} + \mu P_I(\bar{x}) < c^\top x^* + \mu P_I(x^*) = c^\top x^*.$$

If $\bar{x}_I \in \{0,1\}^I$, we have a contradiction to the optimality of x^*. Therefore assume, that $\bar{x}_I \notin \{0,1\}^I$. We then get

$$\begin{aligned} c^\top \bar{x} + \mu P_I(\bar{x}) &\geq l + \mu P_I(\bar{x}) \\ &> l + \frac{u-l}{\varepsilon} P_I(\bar{x}) \\ &> l + u - l \\ &= u \\ &\geq c^\top x^*, \end{aligned}$$

which is a contradiction to our assumptions about $\bar{x}$. Therefore, $c^\top \bar{x} + \mu P_I(\bar{x}) \geq c^\top x^*$. the optimal objective value of Problem 1.1.1 is smaller or equal to the optimal objective value of Problem 5.1.1. □

Unfortunately, in general we do not know the value of $\underline{x}$ and $\bar{x}$ a priori. There are some exceptions, such as the edge formulation of the independent set problem, where all coordinates of all extreme points of the polytope have values in $\{0, 0.5, 1\}$, but for most problems, the value of ε would have to be guessed.

But, these sufficient conditions give additional hints on how to estimate the parameter μ. A nice corollary from this is the following.

Corollary 5.1.4. *For every instance of MILP 1.1.1 there exists a $\mu \in \mathbb{R}_{>0}$, so that any optimal solution of Reformulation 5.1.1 is also an optimal solution of the MILP 1.1.1.*

Proof. By definition, we have $\underline{x} > 0$ and $\bar{x} < 1$. Therefore, there exists a $\varepsilon > 0$ with $\varepsilon < \min\{\underline{x}, 1 - \bar{x}\}$ and from Lemma 5.1.2 we know, that any μ delivers a lower bound and from Lemma 5.1.3 we know that any $\mu > \frac{u-l}{\varepsilon}$ also delivers an upper bound. Therefore, there exists a μ for which the reformulation is exact. □

We now want to give an algorithmic description of how our penalty branch-and-bound method can solve Reformulation 5.1.1. As mentioned before, the reformulation is very similar to the reformulation from Chapter 2, so the algorithm is very similar to Algorithm 2. Therefore, we will use the same notation. We will use $N = (I_0, I_1)$ as a way to identify nodes, where $I_0 \subseteq I$ is the set of indices, we have branched on downwards up to that point in the tree and $I_1 \subseteq I$ is the set of indices, we have branched on upwards. Further, f_N defines the function at node N, i.e.,

$$f_N(x) := c^\top x + \mu \sum_{i \in I_0} x_i + \mu \sum_{i \in I_1} (1 - x_i).$$

Algorithm 6 A First Penalty Branch-and-Bound Algorithm for MILPs

Input: $c \in \mathbb{R}^n$, $P \subseteq \mathbb{R}^n$, $I \subseteq [n]$, $\mu > 0$
Output: A global optimum z^* of Problem 5.1.1.
Set $\mathcal{N} \leftarrow \{(\emptyset, \emptyset)\}$.
Set $f_{\text{inc}} \leftarrow \infty$, $x^*_{\text{inc}} \leftarrow$ none.
while $\mathcal{N} \neq \emptyset$ **do**
  Choose $N = (I_0, I_1) \in \mathcal{N}$ and set $\mathcal{N} \leftarrow \mathcal{N} \setminus \{N\}$.
  Compute $x^*_N \in \operatorname{argmin}\{f_N(x) : x \in P\}$.
  if x^*_N exists **and** $f(x^*_N) < f_{\text{inc}}$ **then**
    Set $x^*_{\text{inc}} \leftarrow x^*_N$ and set $f_{\text{inc}} \leftarrow f(x^*_N)$.
  if $f_N(x^*_N) < f_{\text{inc}}$ **and** there is a $j \in I \setminus (I_0 \cup I_1)$ with $(x^*_N)_j \notin \{0, 1\}$ **then**
    Choose $j \in I \setminus (I_0 \cup I_1)$ with $(x^*_N)_j \notin \{0, 1\}$.
    Set $\mathcal{N} \leftarrow \mathcal{N} \cup \{(I_0 \cup \{j\}, I_1), (I_0, I_1 \cup \{j\})\}$.
return x^*_{inc}

Theorem 5.1.5. *Algorithm 6 terminates after finitely many steps with a global optimal solution of Problem 5.1.1, if the root node is bounded.*

Proof. Let $J_i = \{1, 2\}$ for all $i \in I$ and $g_{i1}(z) = \mu z_i$ and $g_{i2}(z) = \mu(1 - z_i)$. Let $h(z) = c^\top x$ and $X = P$. Now, Problem 5.1.1 is in the form of Problem 3.1.1 and the correctness of Algorithm 6 follows from Theorem 3.1.8. □

5.1.1. Lower and Upper Bounds

We now want to discuss the possibility to find lower and upper bounds for Problem 1.1.1 with our penalty branch-and-bound method. Normally, lower bounds are computed by solving the linear relaxation of the problem, which we have described in Definition 1.1.2. There, the lower bound does not take into account the integrality constraints at all. As shown above, we can also establish lower bounds with our algorithm and an arbitrary μ. This comes in handy, as we cannot determine, how big the penalty parameter μ has to be in order to solve Problem 1.1.1 a priori. We can do something similar in order to find upper bounds for Problem 1.1.1 while not solving the underlying MILP directly. Instead of using a penalty parameter that might be too small, we ignore the original objective function and solve the following problem:

$$\min_{x \in \mathbb{R}^n} \quad \sum_{i \in I} \min\{x_i, 1 - x_i\} \tag{5.1.2a}$$

$$\text{s.t.} \quad x \in P. \tag{5.1.2b}$$

With this problem, we have the penalty reformulation of a feasibility problem as we discussed in Section 1.1.2. Therefore, by solving Problem 5.1.2 with Algorithm 6, we will find a feasible point for Problem 1.1.1, if there is one, or else show, that no such point exists.

5.1.2. Exact Solutions

With the results of the previous section, we can state an algorithm, that provably solves Problem 1.1.1 with our penalty branch-and-bound method. Please note that the way μ is increased can also be done differently, as long as it will become strictly larger withe every iteration.

Algorithm 7 An Exact Penalty Branch-and-Bound Algorithm for MILPs

Input: $c \in \mathbb{R}^n$, $P \subseteq \mathbb{R}^n$, $I \subseteq [n]$
Output: A global optimum x^* of Problem 1.1.1.
Set $\mu \leftarrow 1$, $x^* \leftarrow$ None
while $x^*_I \notin \{0, 1\}^I$ **do**
  Compute $x^* \in \operatorname{argmin}\{c^\top x + \mu \sum_{i \in I} \min\{x_i, 1 - x_i\} \colon x \in P\}$ with Algorithm 6.
  Set $\mu \leftarrow 10\mu$
return x^*

Theorem 5.1.6. *Algorithm 7 terminates after finitely many steps with a global optimal solution of Problem* (1.1.1)*.*

Proof. From Lemma 5.1.2, we know, that the optimal solution of Problem 5.1.1 is a lower bound for every $\mu > 0$, i.e.,

$$c^\top \bar{x} + \mu \sum_{i \in I} \min\{x_i, 1 - x_i\} \leq c^\top x^*$$

with $\bar{x}$ being an optimal solution of Problem 5.1.1 and x^* being an optimal solution of Problem 1.1.1. From Corollary 5.1.4 we know, that there is a finite $\mu > 0$, for which any optimal solution of Reformulation 5.1.1 is also an optimal solution of the MILP 1.1.1. Therefore, after finitely many increments of the parameter μ, we will get a point $\bar{x}$ with $\bar{x}_I \in \{0,1\}^I$. Now, we have

$$c^\top \bar{x} \leq c^\top x^*$$

and $\bar{x}$ has to be optimal for Problem 1.1.1. □

5.2. Further Enhancements of the Method

As we discussed in Sections 2.3 and 4.3, there are various ways to improve the performance of Algorithms 6 and 7. Again, we can try different ways to choose the next index $j \in I$ to branch on, different ways to choose the next subproblem $N \in \mathcal{N}$ to solve, different techniques to warmstart the node problems and different strategies to include valid inequalities. Additionally, we can try different strategies for the choice and increment of the penalty parameter $\mu > 0$.

As the structure of Problem 5.1.1 is very similar to the structure of Problem 2.1.6, we can employ the same strategies. Therefore, as branching rules we will test the random choice of a branching index, choosing the index, for which the corresponding binary constraint is most-violated and the two more sophisticated methods we described in Section 2.3, "pseudocost branching" and the "MIQP-based branching", which will be MILP-based for this problem class. For the node selection strategy we compare breadth-first search and depth-first search with the "lower bound push" strategy described in Section 2.3. For the warmstart, we try to increase the performance by using the optimal basis vector of a node problem as the start basis vector for its child nodes and use the primal simplex to solve node problems. For the valid inequalities we try to include the binary simple cuts, we have already presented in Sections 2.3 and 4.3. Preliminary tests showed that by choosing the initial μ according to Lemma 5.1.3 with $\varepsilon = 10^{-3}$, μ is usually large enough to solve Problem 1.1.1 on the first try. Therefore, we do not investigate more sophisticated methods of choosing and increasing the parameter μ.

5.3. Numerical Results

In this section we present the results of various numerical experiments. In Section 5.3.1, we present the results of numerical experiments we conducted to compare the different enhancement techniques performance-wise. We tested all types of enhancements independently of each other. For very test we take the best settings from previous tests together with "standard" settings for untested parameters. After that, we test our method with the best setting we identified against benchmark approaches with different objectives in Section 5.3.2.

We implemented Algorithm 7 in Python 3.7. All node problems are solved with the LP solver of Gurobi 9.1.2 and all the tests were run on an Intel Xeon CPU E5-2699 v4 @ 2.20 GHz (88 cores) with 756 GB RAM. In this section, we refer to the implementation of Algorithm 7 as MILP-PBB. As most instances were too difficult for our solver, we consider instances that we collected from different versions of the MIPLib for our tests. In particular, we took instances from the MIPLib 2.0 by Bixby et al. (1992), the MIPLib 3.0 by Bixby et al. (1998), the MIPLib 2003 by Achterberg et al. (2006), the MIPLib 2010 by Koch et al. (2011), and the MIPLib 2017 by Gleixner et al. (2021). We restricted the test set to instances without general integer constraints and for versions 3.0 and later we further restricted it to instances with at most 500 binary variables. This resulted in a total of 235 instances.

For the comparisons of parameterizations and algorithms presented in this section we again use logarithmic performance profiles in the sense of Dolan and Moré (2002) as well as tables with the most important statistical measures. For the tables, we aggregated all instances that have been solved by all parameter settings or solution approaches for the specific test w.r.t. the instance size. The first column always states the interval of the dimensions n of the problems. The second column contains the arithmetic mean of node counts respectively running times for all instances solved by every parameterization. The next columns contain the median, the minimum, and the maximum value of the data set. The sixth and seventh column contain the 0.25-quantile, i.e., the node count or running time after which 25 % of instances were solved, as well as the 0.75-quantile. The next two columns contain the geometric mean and the geometric shifted mean. The shift is 100 for the node counts and 10 for the running times. The last column contains the percentage of instances solved to global optimality for the parameterization and instance size. The best value for every measure and instance size among all tables for that test is printed bold. The table of the winning setting, i.e., the best performing parameterization, is included in this section whereas the tables of the other settings are included in Appendix G. In this section, the node count is the total number of needed in all trees that were needed to solve an instance.

The timelimit for these tests is set to 3 h.

5.3.1. Testing the Enhancement Strategies

In this section we conduct a series of tests for the different enhancement techniques discussed in Section 5.2. We start with a test of the different branching rules, followed by a test of

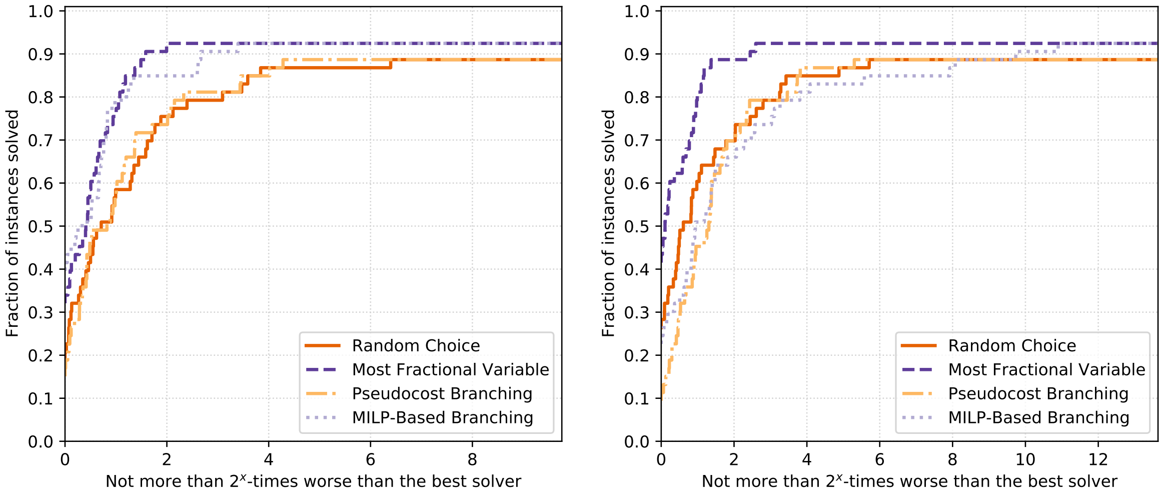


Figure 5.1.: Performance profiles for the number of branch-and-bound nodes (left) and the running time (right) of all branching rules

the node selection strategies, different warm starting techniques, and different strategies for the inclusion of valid inequalities.

The Impact of Different Branching Rules

We first compare the performance of MILP-PBB when equipped with the four different branching rules described in Section 5.2. 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 182 instances from the test set since no parameterization is able to solve them within the time limit. Figure 5.1 displays the performance profiles w.r.t. the required number of branch-and-bound nodes (left figure) and running times (right figure). One can see that the different branching strategies can be divided into two groups in regards to the number of nodes needed. Both the MILP-based branching and the most violated branching solve the most instances and are overall dominant. For the runtime, the picture is a little different. Here, the most violated branching is clearly superior over the MILP-based branching rule as it is less computationally expensive.

Similar conclusions can be drawn from the statistical measures as displayed in Table 5.1 (and Tables G.1–G.3). Here, the MILP-based branching and the most violated branching have the best results for most measures and sizes.

The Impact of Different Node Selection Strategies

We now compare the three node selection strategies described in Section 5.2. To this end, we use the most fractional variable branching strategy, while warmstarts and valid inequalities are disabled. We exclude 183 instances from the set since no parameterization of our method

Table 5.1.: Aggregated nodecounts (top) and runtimes (bottom) for the branching rule test with most-fractional variable branching

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
≤ 200	14802.2	1344.0	45.0	163101.0	48.5	55.5	**1885.2**	14
≤ 500	**7213.0**	**4644.0**	169.0	**22975.0**	170.3	173.0	2058.8	**16**
≤ 2000	25053.8	5271.5	7.0	108761.0	7.2	7.7	2358.4	15
≤ 5000	**586.3**	**267.0**	**29.0**	**1463.0**	**30.2**	**32.6**	**319.8**	**43**
> 5000	**25.5**	19.0	**3.0**	**61.0**	3.1	3.2	23.7	**13**
≤ 200	282.2	6.6	**0.0**	2968.3	**0.0**	**0.0**	**26.7**	14
≤ 500	**56.3**	**39.9**	1.0	**131.7**	1.0	1.0	**29.1**	**16**
≤ 2000	1577.1	53.0	0.1	8404.1	**0.1**	**0.1**	96.3	15
≤ 5000	**27.7**	**10.9**	2.7	**69.4**	2.7	2.8	**17.6**	**43**
> 5000	**193.2**	6.6	1.0	**758.3**	1.0	1.0	28.0	**13**

is able to solve them within the time limit. Based on Figure 5.2, one can notice that again 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. While the depth-first search solves the most instances the fastest, the lower bound push strategy solves slightly more instances overall.

For the statistical measures we present in Tables 5.2, G.4, and G.5 the depth-first search also has a slight advantage for most measures and sizes. But, as the lower bound push solved the most instances, we choose that for our "best-setting" implementation of MILCP-PBB.

Table 5.2.: Aggregated nodecounts (top) and runtimes (bottom) for the node selection test with lower bound push

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
≤ 200	7521.5	1475.0	19.0	82553.0	20.3	23.0	1592.2	14
≤ 500	17443.0	7997.0	**147.0**	69707.0	**148.5**	**151.4**	4911.1	10
≤ 2000	21202.6	4756.0	7.0	121235.0	7.1	7.3	2846.3	**17**
≤ 5000	1378.0	1378.0	41.0	2715.0	47.7	61.1	530.0	40
> 5000	276.4	161.0	51.0	610.0	51.4	52.3	214.2	**16**
≤ 200	86.1	4.4	**0.0**	963.8	**0.0**	**0.0**	13.5	14
≤ 500	279.4	74.7	**1.1**	1262.2	**1.1**	**1.1**	85.3	10
≤ 2000	**1049.4**	88.9	**0.1**	**5947.4**	**0.1**	**0.1**	117.8	**17**
≤ 5000	58.7	58.7	2.8	114.6	3.1	3.6	29.9	40
> 5000	450.5	42.1	19.5	2043.6	19.6	20.0	102.2	**16**

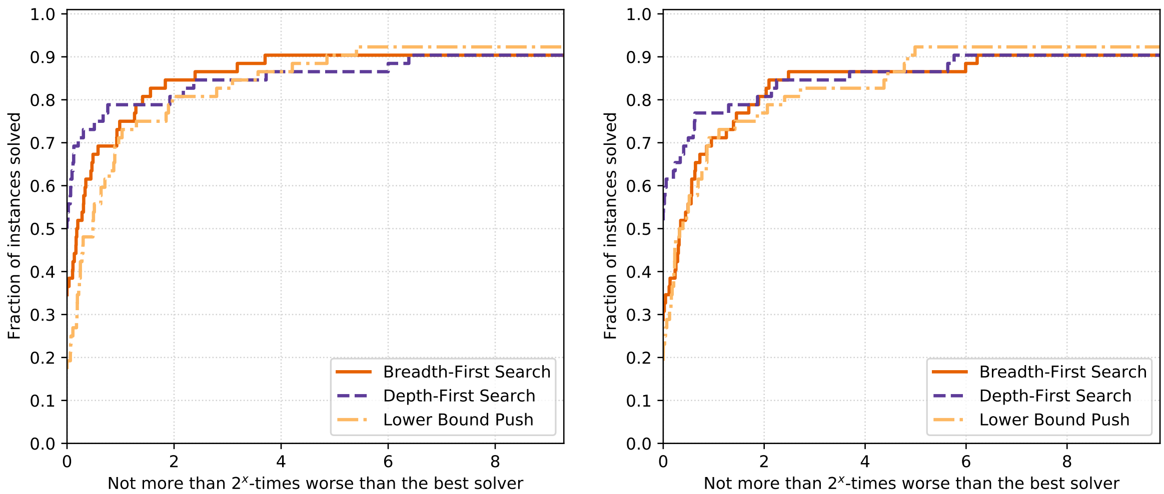


Figure 5.2.: Performance profiles on the number of branch-and-bound nodes (left) and the running time (right) of all node selection strategies.

The Impact of Warmstarts

We now compare the performance of MILP-PBB with and without warmstarts. To this end, we use the most-fractional branching, the lower bound push node selection strategy, and avoid the use of any valid inequalities. We again tried two different techniques within Gurobi to warm start the node problems. First, we used the Gurobi attributes VBasis and CBasis, i.e., we started every node problem with the optimal basis of its parent node. Second, we used the attributes PStart and DStart, where the optimal basis vector of the parent node is computed from the optimal solution. Again, 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 184 instances from the set as no parameterization is able to solve them within the time limit. From Figure 5.3 we can see, that warmstarts do not have a clear impact on the performance. Surprisingly, for the number of nodes needed the use of parameters VBasis and CBasis has a positive impact. Again, the difference 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, effects the overall search tree. For the runtime, the same effect cannot be observed and overall the version of MILP-PBB without warmstarts solves the most instances.

The same can be seen in Tables 5.4, 5.3, and G.6. Both no warmstarts and the warmstarts using parameters VBasis and CBasis dominate for most measures and sizes. Because the warmstarts with VBasis/CBasis have an advantage in the number of nodes needed and the version without warmstarts has an advantage in regards to the number of instances solved overall, we will continue with both versions of MILP-PBB for the following tests.

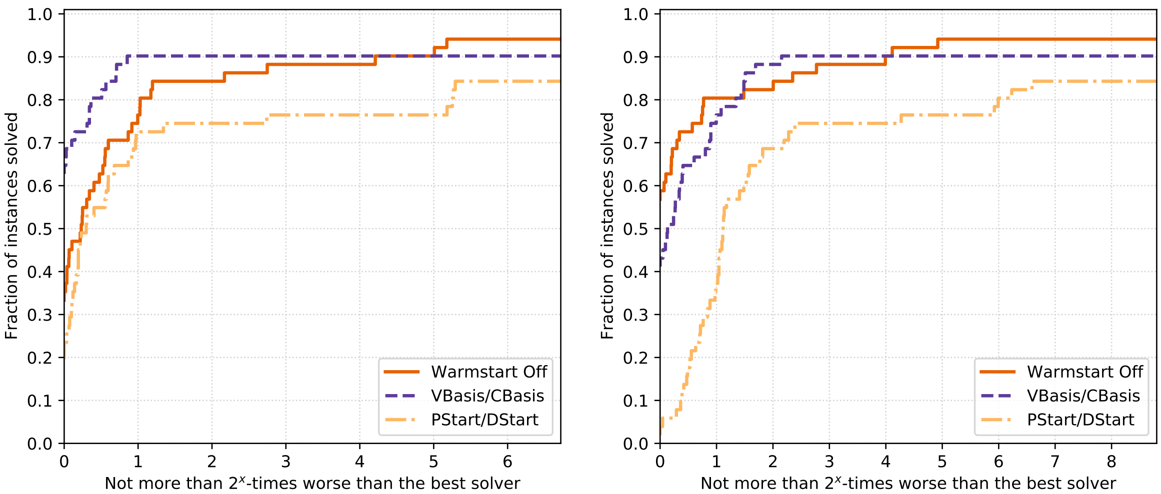


Figure 5.3.: Performance profiles for the number of branch-and-bound nodes (left) and the running time (right) of the warmstart test.

Table 5.3.: Aggregated nodecounts (top) and runtimes (bottom) for the warmstart test with warmstarts off

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
≤ 200	15121.8	1506.0	19.0	153407.0	20.3	23.0	2032.8	14
≤ 500	**17443.0**	7997.0	**147.0**	**69707.0**	**148.5**	**151.4**	4911.1	10
≤ 2000	**30327.2**	3682.5	**7.0**	**113937.0**	26.5	65.5	3459.4	**17**
≤ 5000	1378.0	1378.0	41.0	2715.0	47.7	61.1	530.0	**16**
> 5000	1343.7	441.0	51.0	7583.0	51.6	52.9	436.1	**40**
≤ 200	**161.9**	4.5	**0.0**	**2460.5**	**0.0**	**0.0**	**14.3**	14
≤ 500	**253.6**	77.2	**0.9**	**1135.7**	**0.9**	**0.9**	**80.7**	10
≤ 2000	**1822.6**	**68.7**	**0.1**	**7153.0**	0.5	1.3	133.6	**17**
≤ 5000	57.0	57.0	**3.4**	110.6	**3.7**	**4.2**	30.2	**16**
> 5000	788.8	88.5	20.2	2985.8	20.5	21.0	209.1	**40**

The Impact of Valid Inequalities

We compare a version of MILP-PBB in which all possible simple cuts are added in every node with a version of MILP-PBB in which no simple cuts are added. For this test, the branching rule is set to the most-fractional variable branching rule and the node selection strategy is set to lower bound push. While we wanted to continue with two different versions of MILP-PBB, again the warmstarts using VBasis and CBasis are mutually exclusive with the simple cuts for technical reasons. Hence, we only test the version without warmstarts for this test. 187 instances are excluded for this test.

Table 5.4.: Aggregated nodecounts (top) and runtimes (bottom) for the warmstart test using VBasis/CBasis

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
≤ 200	**14900.4**	**1285.0**	19.0	**149435.0**	20.2	22.7	**1673.8**	**16**
≤ 500	24306.1	**4053.0**	209.0	113939.0	209.9	211.8	**3616.1**	**19**
≤ 2000	31664.5	**2181.0**	**7.0**	122289.0	**8.0**	**10.0**	**1821.7**	**17**
≤ 5000	**735.0**	**735.0**	**31.0**	**1439.0**	**34.5**	**41.6**	**349.0**	10
> 5000	**278.3**	**51.0**	**5.0**	**1129.0**	**5.1**	**5.3**	**147.7**	38
≤ 200	475.3	**3.6**	**0.0**	6662.2	**0.0**	0.1	16.8	**16**
≤ 500	939.3	**52.5**	1.6	5372.9	1.6	1.7	92.5	**19**
≤ 2000	2403.9	**72.0**	**0.1**	9471.6	**0.2**	**0.2**	**107.6**	**17**
≤ 5000	**34.7**	**34.7**	3.7	**65.6**	3.9	**4.2**	**22.2**	10
> 5000	**174.9**	**51.3**	**1.4**	**437.4**	**1.4**	**1.4**	**67.5**	38

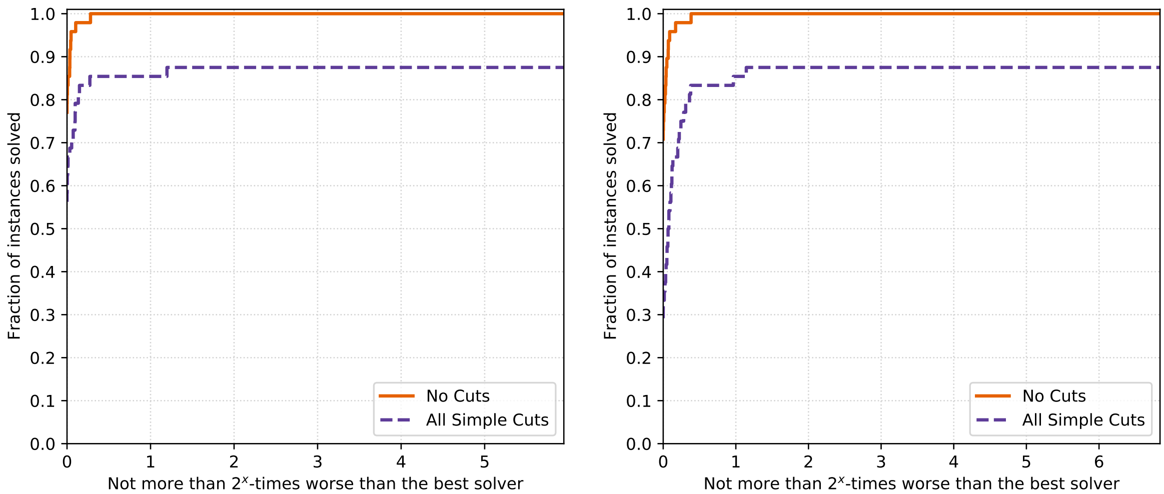


Figure 5.4.: Performance profiles for the number of branch-and-bounds nodes (left) and the running time (right) for variants with all possible simple cuts and without any.

As can be seen in Figure 5.4, incorporating the simple cuts has a negative impact both on the number of branch-and-bound nodes as well as on the running time and results in less instances solved overall. This is also obvious from the results in Tables G.7 and 5.5. For most measures and instance sizes, the approach without the simple cuts outperforms the method with all cuts both w.r.t. the node counts and the running times.

5.3.2. Various Benchmark Tests

We know want to compare our best-setting variants of MILP-PBB against Gurobi. For a fairer comparison, we disabled cutting planes, heuristics, and presolve for Gurobi. Further,

Table 5.5.: Aggregated nodecounts (top) and runtimes (bottom) for the valid inequalities test without cuts

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
≤ 200	**14468.4**	**1497.0**	**19.0**	**153407.0**	**20.4**	**23.2**	**1997.4**	**40**
≤ 500	25922.0	**16917.0**	**147.0**	69707.0	**187.5**	**268.6**	7151.8	**10**
≤ 2000	44314.9	**9567.0**	**7.0**	**169097.0**	46.0	124.1	**8199.4**	**14**
≤ 5000	**1378.0**	**1378.0**	**41.0**	**2715.0**	**47.7**	**61.1**	**530.0**	**16**
> 5000	**1194.4**	**301.0**	**51.0**	**7583.0**	**51.7**	53.2	**387.1**	**17**
≤ 200	**350.2**	**5.0**	**0.0**	5272.4	**0.0**	**0.0**	**22.0**	**40**
≤ 500	533.5	347.9	**1.5**	1436.8	**1.9**	**2.7**	151.1	**10**
≤ 2000	2467.3	**154.8**	**0.1**	7876.2	0.9	2.3	**347.1**	**14**
≤ 5000	**58.0**	**58.0**	3.1	**113.0**	3.4	**3.9**	**30.1**	**16**
> 5000	**1186.6**	370.2	26.0	**3233.9**	26.2	26.8	**333.8**	**17**

we restricted both the LP solver for the nodes in MILP-PBB and the MILP solver of Gurobi to a single thread. We first evaluate possibilities two find better lower and upper bounds on the objective value of the instances. Afterward, we also want to compare ourselves to the benchmark in actually finding optimal solutions of the problems. The best-setting methods use the most-fractional branching rule, the lower bound push, and no simple cuts. For the warmstarts we test both the version without warmstarts (called MILP-PBB 1) and the version using VBasis/CBasis (called MILP-PBB 2).

Finding Lower Bounds

As we can conclude from Lemma 5.1.2, the optimal objective value of Reformulation 5.1.1 for any $\mu > 0$ is a lower bound for the corresponding MILP. We can therefore use the reformulation to compute lower bound by choosing a small parameter μ. From the numerical experiences we gained in the experiments described in Section 2.4 and Appendix A, one can expect that a smaller parameter μ leads to a better performance. Therefore, we set $\mu = 0.1$ for this test and do not repeat the branch-and-bound algorithm for a higher μ if we do not find the optimal solution of the MILP directly. We compared this lower bound to the lower bound given by the linear relaxation of the problem. Obviously, it is much faster to solve the relaxation, as it is done in the root node of our method. Hence, it is not sensible to compare the runtime or number of nodes needed for the two procedures. Instead we only compare the computed lower bounds for the instances, on which at least one of the two MILP-PBB versions did not run into the time limit. Unfortunately, the lower bounds on all instances were identical within numerical exactness, as the penalization of non-integrality was not high enough, and we can conclude, that MILP-PBB in its current form is not suitable to compute lower bounds.

Finding Upper Bounds

In Section 5.1.1 we mentioned, that we can use our method to compute upper bounds on the optimal objective value of MILPs with our method. To achieve this, we set the objective function of the MILP to 0. Hence, the objective function of Reformulation 5.1.1 only consists of the penalty term P_I. We compare our method to the MILP solver of Gurobi. 27 instances were excluded, as no solver was able to solve them.

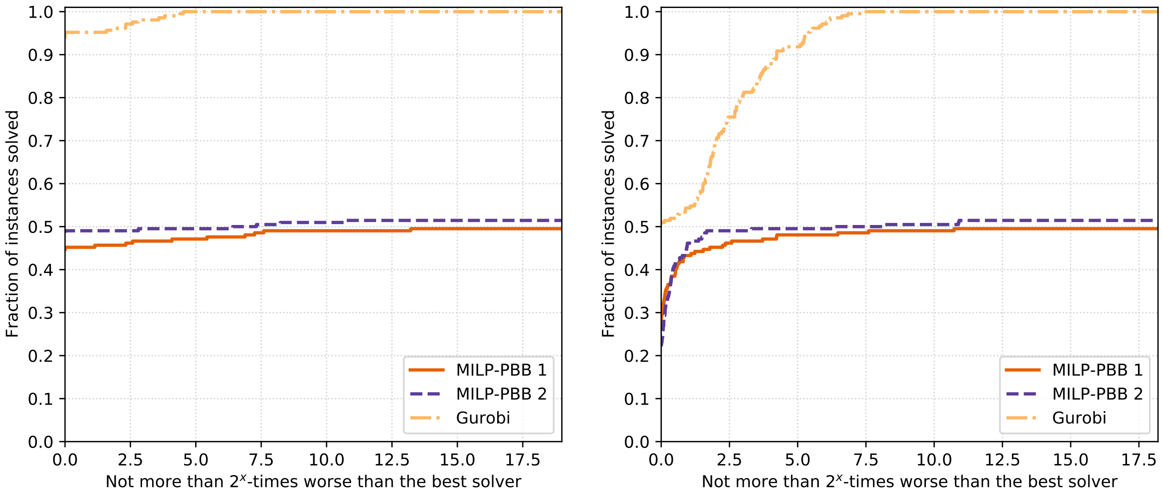


Figure 5.5.: Performance profiles for the number of branch-and-bounds nodes (left) and the running time (right) for the upper bound test.

From Figure 5.5 it is obvious, that Gurobi clearly outperforms our method in regards to the number of nodes needed and the number of instances solved overall. For the runtime, the picture is less clear and our method is competitive for quite a few instances. But the observations from the performance profiles are put into perspective by the results in Tables 5.6, G.8, and G.9. Here it is obvious that for the majority of instances a feasible point was found within the root node, which makes the results w.r.t. the runtime become less relevant.

While runtime and node count appear to be not very interesting and important for this test, we can notice significant differences for the quality of the bounds the different methods produce. To compare the bounds, we again use a logarithmic performance profile which can be seen in Figure 5.6.

Our methods produce better bounds for quite a few instances that were solved by our method than the bounds Gurobi found. But as Gurobi had the better performance overall and solved more instances, it also found better bounds for a lot more instances in total. Unfortunately, we again have to conclude, that MILP-PBB in its current form is not suitable to compute upper bounds, especially considering that there are even better and faster heuristics to find feasible points for mixed-integer problems than Gurobi, such as rounding or feasibility pump methods.

Table 5.6.: Aggregated nodecounts (top) and runtimes (bottom) for the upper bound test for Gurobi

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
≤ 200	**1.4**	**1.0**	**1.0**	**5.0**	**1.0**	**1.0**	**1.4**	**81**
≤ 500	1.4	**1.0**	**1.0**	6.0	**1.0**	**1.0**	1.4	**90**
≤ 2000	**1.1**	**1.0**	**1.0**	**3.0**	**1.0**	**1.0**	**1.1**	**96**
≤ 5000	11.7	**1.0**	**1.0**	65.0	**1.0**	**1.0**	9.6	**95**
> 5000	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**83**
≤ 200	0.3	**0.0**	**0.0**	3.1	**0.0**	**0.0**	0.3	**81**
≤ 500	0.8	0.1	**0.0**	7.3	**0.0**	**0.0**	0.7	**90**
≤ 2000	0.2	0.1	**0.0**	1.1	**0.0**	**0.0**	0.2	**96**
≤ 5000	0.9	0.8	0.2	1.8	0.2	0.2	0.9	**95**
> 5000	18.3	8.7	0.6	132.2	0.6	0.6	10.6	**83**

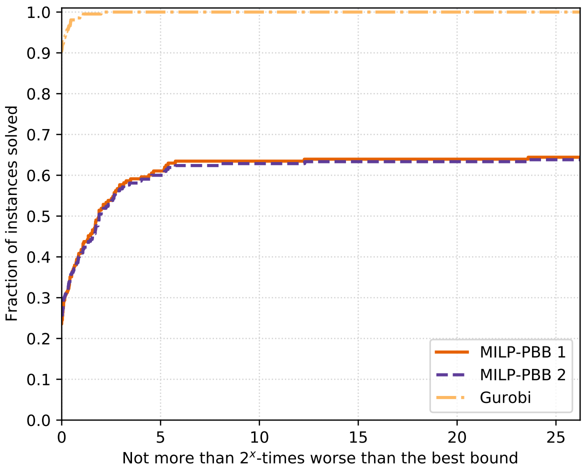


Figure 5.6.: Performance profiles for the upper bound provided by the different methods in the upper bound test.

Finding Exact Solutions

We now want to compare ourselves to Gurobi in solving the instances from before to global optimality. 122 instances were excluded, as no solver was able to solve them. It is very obvious from Figure 5.7 that our method is clearly outperformed by Gurobi, which is probably not surprising considering the nature of Gurobi as the state-of-the-art commercial solver for MILPs. This is also apparent from Tables 5.7, G.10, and G.11.

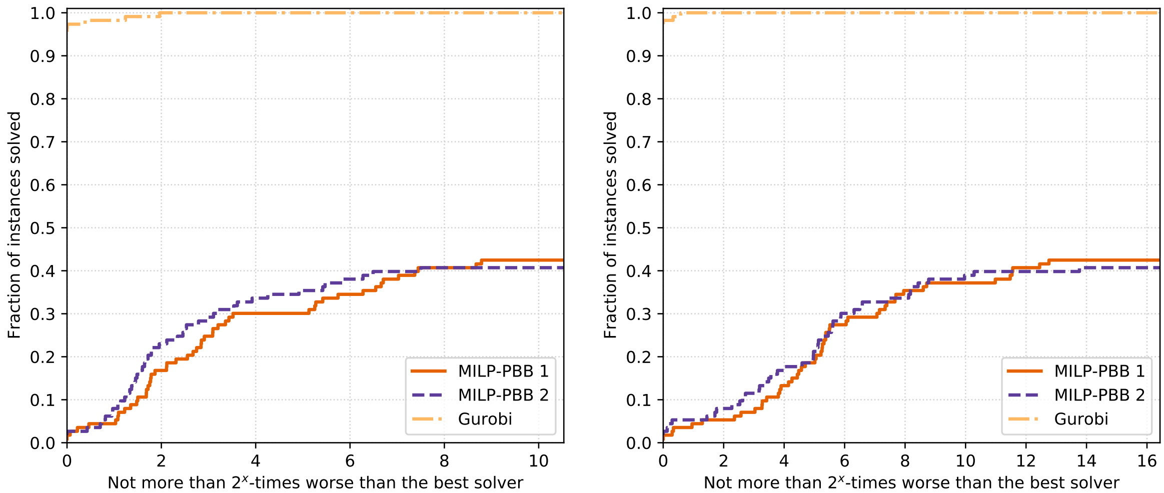


Figure 5.7.: Performance profiles for the number of branch-and-bounds nodes (left) and the running time (right) for the benchmark test.

Table 5.7.: Aggregated nodecounts (top) and runtimes (bottom) for the benchmark test for Gurobi

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
≤ 200	**3379.0**	**217.5**	**9.0**	**55082.0**	**9.7**	**11.0**	**398.0**	**56**
≤ 500	**3760.0**	**400.0**	**23.0**	**24277.0**	**23.2**	**23.5**	**490.4**	**33**
≤ 2000	**1821.5**	**36.0**	**3.0**	**10554.0**	**3.0**	**3.0**	**206.2**	**35**
≤ 5000	**378.5**	**378.5**	**12.0**	**745.0**	**13.8**	**17.5**	**207.6**	**66**
> 5000	**275.0**	**14.5**	**3.0**	1133.0	**3.0**	**3.1**	**101.0**	**41**
≤ 200	**1.5**	**0.1**	**0.0**	**27.2**	**0.0**	**0.0**	**0.8**	**56**
≤ 500	**5.3**	**1.1**	**0.2**	**32.8**	**0.2**	**0.2**	**3.1**	**33**
≤ 2000	**9.4**	**0.3**	0.2	**54.0**	0.2	0.2	**4.2**	**35**
≤ 5000	**1.8**	**1.8**	**0.6**	**2.9**	**0.6**	**0.6**	**1.7**	**66**
> 5000	**70.1**	**37.8**	**2.1**	**237.9**	**2.1**	**2.1**	**33.9**	**41**

Chapter 6

Conclusion

In this thesis we have presented and investigated a novel type of branch-and-bound algorithms. While this algorithm class is closely related to classic branch-and-bound methods, the principles of how the algorithms work are fundamentally different. Classic branch-and-bound methods are methods, where a problematic, usually non-convex set of constraints is solved by exploiting the combinatorial nature of these constraints. Usually, these constraints are binary or integer constraints and branch-and-bound methods are the state-of-the-art for solving mixed-integer optimization problems. The core of these methods is a divide-and-conquer approach that deals with the problematic constraints one after another. We have taken this core principle and turned it into a solution method for a different set of problem classes. The problems we deal with have problematic, non-convex terms in the objective function instead of problematic constraints. While classic branch-and-bound methods deal with the constraints by dividing the feasible set of a problem into subsets by successively adding constraints, we do not change the feasible set. Instead, as the problematic terms are part of the objective function, we successively alter the objective function of the problem by the addition of non-negative terms.

We have described the principles of this novel penalty branch-and-bound method and analyzed the algorithm for different problem classes, for which the method is applicable. For the specific problem classes we investigated we pointed out different problem specific possibilities to enhance the performance of the method and tested them numerically. For both monotone and non-monotone MILCP, we were able to show that the method is not only very well performing compared to different benchmark approaches, but that it also has other advantages. First, the method does not rely on big-M constants as many of the existing solution methods. Secondly, our method not only finds solutions of the investigated MILCP, but also finds points, that minimize a measure of infeasibility, if there is no feasible solution.

We also investigated possibilities to solve problems from the broad class of MILP. While not being particularly successful when we look at the numerical experiments, we were able to present a completely novel approach for solving MILP by using penalty reformulations of the problem. We also analyzed different ways to enhance the performance and showed ways to not only compute solutions of MILP, but also upper and lower bounds.

Besides investigating the applicability of our method for specific and known problem classes, we also explored generalizations of problem classes that could be solved with our method theoretically. We found two different generalizations. One is the class of problems with a convex feasible set and an objective function consisting of the sum of minimum functions over a set of convex functions. The second is a class of problems that have a convex feasible set and an objective function with a sum of piecewise convex functions. For both classes we presented an algorithm to solve them and proved its correctness.

Despite these contributions, there is still room for future work and interesting questions remain open. One big strength of classic branch-and-bound methods are the different possibilities to tailor the method to different problems. There are a lot of different classes of valid inequalities, branching rules, node selection strategies, presolve techniques, and heuristics that help the performance of the algorithms. While we were successful in finding similar techniques to improve the performance of the penalty branch-and-bound algorithm, there is still a lot to explore in order to further improve the performance.

Additionally, the instances we tested for MILCPs were created synthetically. As there are a lot of practical problems that utilize MILCP, it would be interesting to test our method on real-world problem instances.

Lastly, we have to say that there is a large discrepancy between the generalized problem classes we identified and the specific problem classes we investigated in detail. While the generalized problem classes are very big, they are not classes that appear naturally and in this general form they are probably too difficult to solve in practice. The specific problem classes do appear in applications, but are a rather specific classes. It would be therefore very interesting to identify other problem classes that appear in applications and that can be solved with our method in practice.

Overall, this novel type of algorithm provides exciting opportunities for the future of non-convex mathematical optimization.

Appendix A

Detailed Description of the Test Set for Monotone MILCP

In order to build a proper test set of MILCP instances, we created matrices $M \in \mathbb{R}^{n \times n}$ using the sprandsym function of MATLAB for sizes

$$n \in \{50, 100, 150, 200, 250, 300, 350, 400, 450, 500\}.$$

For a first preliminary test set, the corresponding matrix densities have been chosen so that they roughly follow the sigmoid-like function

$$d(n) := \frac{1}{1 + e^{\frac{1}{50}n - 5}}.$$

Moreover, we obtain a random non-negative spectrum with an upper bound of 100 for the eigenvalues. The set of integer variables I has been chosen as a random sample of size

$$r(n) := \frac{1}{5\left(1 + e^{\frac{1}{80}n - 3}\right)}.$$

Finally, we built vectors $q \in \mathbb{R}^n$ for the four different "degrees of feasibility"; see Section 2.4. 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 A.1, 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 fraction of integer variables to make the instances of Type (a)–(c) comparably difficult. Instances of Type (a) have 8 % integer variables, instances of Type (b) have 4 % integer variables, and instances of Type (c) have 10 % integer variables.

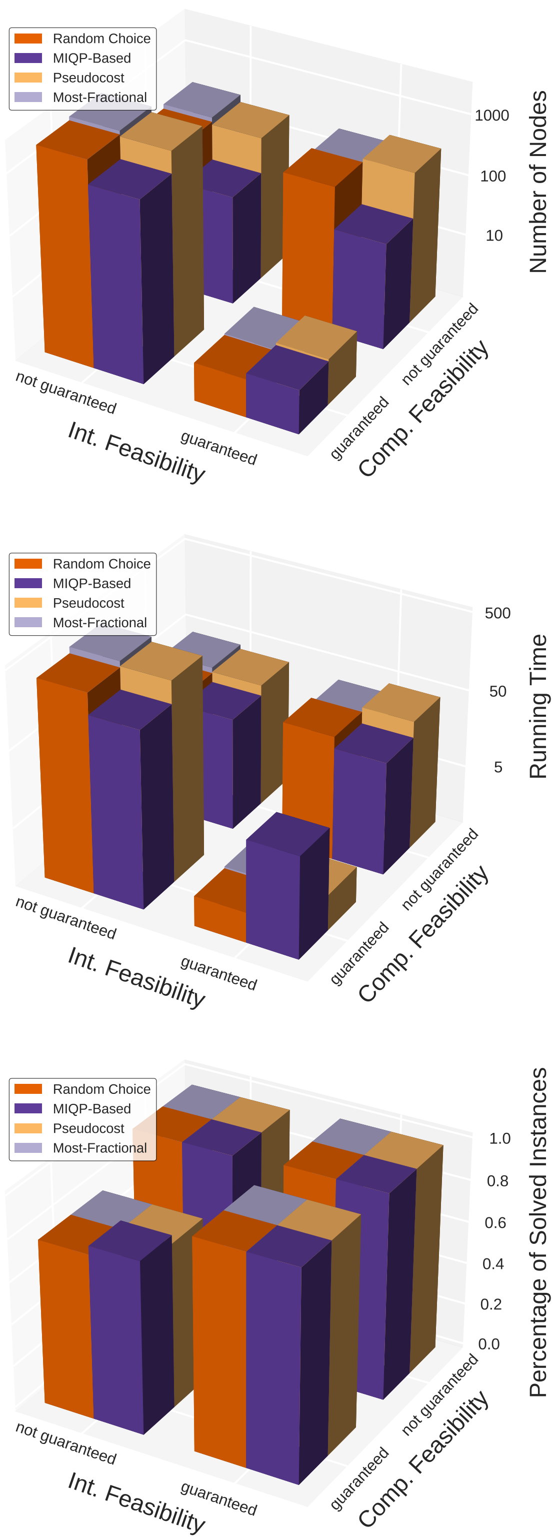


Figure A.1.: Test of different branching rules in dependence of the “degree of feasibility” of the instance

Appendix **B**

Tables of Aggregated Results of Other Settings for Monotone MILCP

In what follows, we include all tables for the aggregated running times and node counts of the settings not reported in Section 2.4.

B.1. Branching Rule Test

Table B.1.: Aggregated node counts (top) and runtimes (bottom) for the branching rule test with random choice

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
50	**13.8**	**8.0**	**3.0**	39.0	**5.0**	**21.0**	**13.3**	**100**
100	18.1	14.0	**3.0**	83.0	**7.0**	26.0	17.2	**100**
150	40.5	37.0	**7.0**	155.0	17.0	44.5	37.1	**100**
200	69.3	60.0	**15.0**	269.0	34.0	95.5	63.7	**100**
250	142.3	118.0	9.0	399.0	79.5	196.5	122.0	**100**
300	335.5	224.0	17.0	1281.0	159.0	357.5	261.5	**100**
350	704.7	435.0	**7.0**	3499.0	128.0	927.0	427.0	93
400	620.4	418.0	65.0	2221.0	115.5	1061.0	407.0	70
450	630.5	473.0	73.0	2399.0	203.0	946.0	464.2	53
500	791.9	715.0	105.0	1479.0	471.0	1151.0	644.6	27
50	0.4	0.2	**0.1**	1.2	**0.1**	0.4	0.4	**100**
100	2.7	1.6	0.6	19.4	**0.9**	3.1	2.4	**100**
150	24.0	19.6	2.4	100.0	6.6	25.5	18.0	**100**
200	72.6	56.6	**13.1**	248.0	37.4	105.3	60.1	**100**
250	213.6	184.3	12.6	561.1	126.1	279.8	160.7	**100**
300	552.3	442.0	**45.0**	**1494.0**	300.4	653.0	424.0	**100**
350	1097.6	685.5	38.0	3240.6	326.9	1690.4	717.6	93
400	1105.3	1052.3	224.6	2848.0	386.3	1768.1	823.7	70
450	1327.8	1160.7	261.9	3215.6	636.7	1878.1	1074.1	53
500	1502.6	1609.6	394.5	2167.7	1201.0	1972.1	1339.9	27

Table B.2.: Aggregated node counts (top) and runtimes (bottom) for the branching rule test with pseudocost branching

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
50	15.1	9.0	**3.0**	43.0	**5.0**	26.5	14.5	**100**
100	18.5	16.0	**3.0**	**63.0**	**7.0**	25.0	17.7	**100**
150	41.3	38.0	**7.0**	147.0	15.5	**44.0**	37.8	**100**
200	73.4	51.0	17.0	287.0	34.0	97.0	66.0	**100**
250	142.9	104.0	11.0	421.0	75.5	194.0	120.0	**100**
300	345.7	235.0	19.0	**1171.0**	147.5	390.0	264.0	**100**
350	704.0	389.0	9.0	3041.0	115.0	958.0	422.5	97
400	658.7	395.0	67.0	2513.0	86.5	1090.5	394.9	67
450	620.5	377.0	53.0	2329.0	244.0	916.0	453.6	53
500	750.4	591.0	107.0	1395.0	526.0	1054.0	625.3	23
50	0.3	0.2	**0.1**	0.9	0.2	0.4	0.3	**100**
100	2.5	**1.5**	**0.4**	15.0	**0.9**	3.1	**2.2**	**100**
150	24.0	19.8	**2.3**	98.4	6.5	25.7	18.1	**100**
200	72.9	55.3	14.5	250.1	33.3	95.6	58.6	**100**
250	211.8	180.3	18.0	519.1	113.6	273.7	155.3	**100**
300	594.2	462.7	60.2	1607.5	310.0	767.7	450.4	**100**
350	1178.3	997.9	40.9	3177.6	312.5	1862.6	741.4	97
400	1185.6	914.0	251.3	3495.7	311.3	1879.6	817.2	67
450	1258.5	901.8	**223.1**	3084.2	598.5	1902.2	1009.1	53
500	1473.1	1581.2	**352.4**	2133.3	1257.9	1864.7	1302.6	23

Table B.3.: Aggregated node counts (top) and runtimes (bottom) for the branching rule test with most-fractional variable branching

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
50	13.9	**8.0**	**3.0**	**37.0**	**5.0**	22.5	13.4	**100**
100	18.6	16.0	**3.0**	81.0	**7.0**	23.0	17.8	**100**
150	41.3	**33.0**	**7.0**	151.0	**14.0**	46.0	37.8	**100**
200	67.9	58.0	23.0	**233.0**	35.0	90.5	62.9	**100**
250	144.5	126.0	7.0	403.0	77.5	163.0	124.3	**100**
300	354.7	228.0	19.0	1385.0	139.0	391.0	263.2	**100**
350	742.2	439.0	**7.0**	3641.0	124.0	1001.0	445.6	90
400	631.0	434.0	51.0	2343.0	119.0	1038.5	412.1	63
450	648.9	539.0	93.0	2265.0	224.0	908.0	483.8	57
500	711.9	683.0	119.0	1375.0	429.0	974.0	585.0	27
50	**0.2**	**0.1**	**0.1**	0.8	**0.1**	**0.2**	**0.2**	**100**
100	2.6	**1.5**	**0.4**	16.5	**0.9**	3.1	2.3	**100**
150	23.4	18.3	2.8	82.8	6.2	25.7	17.8	**100**
200	68.1	51.8	22.4	214.4	31.5	80.8	57.1	**100**
250	221.7	214.7	**12.4**	584.8	128.8	258.4	167.0	**100**
300	592.6	456.9	53.9	1763.6	270.0	728.6	437.6	**100**
350	1224.6	956.6	**32.2**	3573.5	352.4	2046.3	786.4	90
400	1202.0	1106.9	197.3	3345.7	401.1	1506.8	882.5	63
450	1343.3	1298.9	329.6	3057.6	551.8	2039.7	1085.8	57
500	1390.6	1469.6	443.9	2127.8	1097.6	1749.0	1243.3	27

B.2. Node Selection Test

Table B.4.: Aggregated node counts (top) and runtimes (bottom) for the node selection test with breadth-first search

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
50	**13.9**	**8.0**	**3.0**	**37.0**	**5.0**	**21.0**	**13.3**	**100**
100	**16.1**	**10.0**	**3.0**	**73.0**	**7.0**	**21.0**	**15.3**	**100**
150	**35.6**	**24.0**	**7.0**	**139.0**	**13.5**	**42.5**	**32.6**	**100**
200	**59.5**	**43.0**	**15.0**	**235.0**	**31.0**	**82.5**	**54.9**	**100**
250	**112.3**	**96.0**	**5.0**	**353.0**	**67.5**	**131.0**	**97.6**	**100**
300	200.3	142.0	**15.0**	743.0	**95.0**	**254.5**	164.4	**100**
350	277.1	**144.0**	**7.0**	**735.0**	**79.0**	**479.5**	203.2	**77**
400	351.3	**271.0**	**47.0**	**813.0**	**78.0**	**613.0**	259.4	67
450	315.1	345.0	**51.0**	651.0	127.0	**473.5**	264.1	**47**
500	461.0	519.0	**71.0**	**923.0**	**265.5**	544.5	381.6	20
50	**0.3**	**0.2**	0.2	0.7	**0.2**	**0.3**	**0.3**	**100**
100	4.7	3.0	0.8	23.8	2.0	**4.7**	4.1	**100**
150	**25.9**	19.0	6.0	**90.3**	10.3	**30.6**	**21.1**	**100**
200	**91.2**	73.2	30.4	**272.7**	54.4	115.7	79.3	**100**
250	**249.2**	228.0	37.1	807.3	**148.4**	295.5	**198.8**	**100**
300	595.8	475.8	**101.4**	1993.4	281.5	826.4	467.3	**100**
350	1103.9	576.6	110.1	3052.6	**364.4**	1647.0	742.1	**77**
400	**1452.9**	1173.7	**304.9**	3312.1	515.5	**2336.5**	1077.2	67
450	1599.1	1707.9	418.3	2893.0	763.8	2440.0	1328.6	**47**
500	**2000.0**	**2069.0**	**535.6**	3257.8	1530.5	**2555.6**	**1742.3**	20

Table B.5.: Aggregated node counts (top) and runtimes (bottom) for the node selection test with lower-bound-push

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
50	**13.9**	**8.0**	**3.0**	**37.0**	**5.0**	**21.0**	**13.3**	**100**
100	**16.1**	**10.0**	**3.0**	**73.0**	**7.0**	**21.0**	**15.3**	**100**
150	**35.6**	**24.0**	**7.0**	**139.0**	**13.5**	**42.5**	**32.6**	**100**
200	**59.5**	**43.0**	**15.0**	**235.0**	**31.0**	**82.5**	**54.9**	**100**
250	**112.3**	**96.0**	**5.0**	**353.0**	**67.5**	**131.0**	**97.6**	**100**
300	195.5	142.0	**15.0**	**741.0**	**95.0**	**254.5**	162.6	**100**
350	**277.0**	**144.0**	**7.0**	**735.0**	**79.0**	**479.0**	**203.1**	73
400	351.3	**271.0**	**47.0**	**813.0**	**78.0**	**613.0**	259.4	63
450	**314.6**	**343.0**	**51.0**	**647.0**	127.0	**473.5**	**263.7**	**47**
500	**459.3**	**516.0**	**71.0**	**923.0**	**265.5**	**540.0**	**380.3**	20
50	**0.3**	**0.2**	0.2	0.6	**0.2**	**0.3**	**0.3**	**100**
100	4.6	2.7	**1.2**	**20.6**	**1.9**	5.0	4.1	**100**
150	29.0	20.5	**5.7**	97.2	**10.2**	34.3	23.1	**100**
200	99.3	75.6	**25.7**	329.1	54.0	131.7	83.9	**100**
250	267.2	**217.4**	**34.8**	872.2	158.8	307.6	208.1	**100**
300	630.8	495.9	116.9	2252.5	350.4	**761.6**	501.2	**100**
350	1233.5	606.3	105.0	3389.2	410.1	1977.7	816.9	73
400	1594.7	1605.4	337.1	3521.9	556.6	2717.4	1179.8	63
450	1573.3	1729.7	399.1	2829.8	783.4	**2206.1**	1322.0	**47**
500	2161.2	2345.8	649.4	3359.0	**1346.4**	3027.0	1862.0	20

B.3. Warmstart Test

Table B.6.: Aggregated node counts (top) and runtimes (bottom) for the warmstart test without any warmstart

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
50	**13.9**	**8.0**	**3.0**	**37.0**	**5.0**	**21.0**	**13.3**	**100**
100	**16.1**	**10.0**	**3.0**	**73.0**	**7.0**	**21.0**	**15.3**	**100**
150	35.6	**24.0**	**7.0**	**139.0**	13.5	**42.5**	32.6	**100**
200	**59.5**	**43.0**	**15.0**	**235.0**	**31.0**	**82.5**	54.9	**100**
250	**112.3**	**96.0**	**5.0**	**353.0**	**67.5**	**131.0**	**97.6**	**100**
300	**193.3**	**141.0**	**15.0**	**741.0**	**95.0**	**254.5**	161.7	**100**
350	356.3	**215.0**	**7.0**	1119.0	**79.0**	**679.0**	251.1	**87**
400	418.3	**279.0**	**47.0**	**1245.0**	**81.0**	**719.0**	301.1	70
450	314.7	**345.0**	51.0	**647.0**	**126.5**	**473.5**	263.7	47
500	478.6	**543.0**	71.0	**1043.0**	**189.0**	**547.0**	372.3	23
50	**0.3**	**0.3**	0.2	1.2	**0.2**	**0.3**	**0.3**	**100**
100	4.5	2.4	1.1	20.6	1.9	5.4	4.0	**100**
150	26.8	19.3	6.5	111.9	10.0	31.4	21.5	**100**
200	90.4	62.8	28.7	295.1	50.6	116.2	77.5	**100**
250	241.2	209.7	32.3	762.3	161.0	309.2	190.7	**100**
300	533.8	451.4	112.5	1764.6	260.8	655.1	429.0	**100**
350	1256.3	787.4	105.2	3360.6	369.7	2311.8	826.9	**87**
400	1550.7	1401.0	300.9	3522.8	**396.9**	2497.7	1125.9	70
450	1460.0	**1505.9**	345.3	2608.6	751.1	2112.5	1223.4	47
500	**1995.4**	2206.2	584.4	**3375.5**	1055.1	**2755.9**	**1663.8**	23

Table B.7.: Aggregated node counts (top) and runtimes (bottom) times for the warmstart test using PStart/DStart

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
50	**13.9**	**8.0**	**3.0**	**37.0**	**5.0**	**21.0**	**13.3**	**100**
100	**16.1**	**10.0**	**3.0**	**73.0**	**7.0**	**21.0**	**15.3**	**100**
150	**35.3**	**24.0**	**7.0**	**139.0**	**12.0**	**42.5**	**32.3**	**100**
200	**59.5**	**43.0**	**15.0**	**235.0**	**31.0**	**82.5**	**54.8**	**100**
250	**112.2**	**96.0**	**5.0**	**353.0**	**67.5**	**131.0**	**97.6**	**100**
300	**193.3**	**141.0**	**15.0**	**741.0**	**95.0**	**254.5**	**161.6**	**100**
350	**355.4**	**215.0**	**7.0**	**1107.0**	**79.0**	**679.0**	**249.9**	83
400	**417.6**	**279.0**	**47.0**	**1245.0**	**81.0**	**719.0**	**300.3**	70
450	**312.4**	**345.0**	**41.0**	**649.0**	**126.5**	**473.5**	**258.6**	47
500	**476.6**	**543.0**	**61.0**	**1043.0**	**189.0**	**547.0**	**366.6**	17
50	**0.3**	**0.3**	**0.1**	1.2	**0.2**	**0.3**	**0.3**	**100**
100	4.0	2.4	1.3	15.7	1.8	4.5	3.6	**100**
150	25.1	17.9	4.5	84.9	**9.3**	**27.7**	20.1	**100**
200	86.2	63.2	25.9	296.0	48.5	116.0	73.5	**100**
250	245.0	207.1	**26.5**	754.8	145.4	309.0	189.6	**100**
300	549.8	450.0	103.4	1586.1	299.5	745.0	442.9	**100**
350	1324.4	**708.9**	113.3	3361.2	383.7	2404.4	857.4	83
400	1544.5	**1039.5**	292.4	3201.7	460.1	2421.4	1136.5	70
450	1500.2	1591.4	406.3	2919.3	719.5	2102.1	1262.7	47
500	2119.3	**2194.8**	838.8	3386.6	**1015.6**	3160.8	1822.6	17

B.4. Valid Inequalities Test

Table B.8.: Aggregated node counts (top) and runtimes (bottom) for the valid inequalities test with no cuts

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
50	13.9	8.0	**3.0**	37.0	**5.0**	21.0	13.3	**100**
100	16.1	10.0	**3.0**	73.0	7.0	21.0	15.3	**100**
150	35.6	24.0	**7.0**	139.0	13.5	42.5	32.6	**100**
200	59.5	43.0	15.0	235.0	31.0	82.5	54.9	**100**
250	113.1	99.0	**5.0**	353.0	67.0	131.0	98.0	**100**
300	193.3	141.0	15.0	741.0	95.0	254.5	161.7	**100**
350	532.9	309.0	7.0	2701.0	81.0	735.0	334.5	**100**
400	860.0	579.0	47.0	3929.0	109.0	1056.0	493.6	90
450	593.0	402.0	51.0	1713.0	163.5	834.5	416.3	67
500	816.1	923.0	71.0	1523.0	519.0	1107.0	663.9	37
50	**0.2**	**0.2**	**0.1**	0.5	**0.2**	**0.3**	**0.2**	**100**
100	2.3	1.6	0.8	10.2	1.2	2.1	2.2	**100**
150	13.8	10.0	3.3	50.6	5.8	16.0	11.7	**100**
200	43.4	31.3	13.0	150.8	24.2	58.1	37.4	**100**
250	136.4	111.7	14.2	444.8	74.5	179.0	102.4	**100**
300	293.8	244.4	45.6	851.7	150.1	391.9	235.9	**100**
350	788.0	530.1	64.5	1962.8	212.7	1342.9	527.6	**100**
400	1250.2	1140.1	192.5	3514.5	364.3	1476.5	867.6	90
450	1215.6	1032.2	252.7	2872.7	445.4	1917.3	934.0	67
500	1545.5	1397.8	363.5	2646.2	1043.1	2135.5	1345.9	37

B.5. A First Benchmark Test

Table B.9.: Aggregated node counts (top) and runtimes (bottom) for the first benchmark test for the MILP reformulation

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
50	36.9	24.0	5.0	142.0	15.0	46.8	33.7	**100**
100	330.6	193.0	13.0	1416.0	69.5	462.5	227.6	**100**
150	866.5	921.0	26.0	2928.0	167.8	1179.8	582.3	**100**
200	6122.2	2985.0	122.0	78700.0	1454.8	4958.8	2755.0	**100**
250	30406.6	11944.0	287.0	185996.0	5482.0	29671.0	12745.4	83
300	36647.2	45035.0	2065.0	71988.0	21157.5	46958.8	24581.9	27
50	**0.1**	**0.1**	0.1	**0.2**	**0.1**	**0.2**	**0.1**	**100**
100	0.7	0.4	0.2	1.7	0.3	0.9	0.7	**100**
150	2.9	3.2	**0.3**	7.3	0.9	4.2	2.8	**100**
200	64.1	28.3	1.3	941.0	10.7	46.1	28.8	**100**
250	599.2	239.2	2.6	3169.1	106.1	572.8	252.7	83
300	1434.6	1305.0	56.4	3529.4	809.4	1797.3	926.1	27

B.6. A Second Benchmark Test

Table B.10.: Aggregated node counts (top) and runtimes (bottom) for the second benchmark test for the MIQP reformulation

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
100	45.1	33.5	11.0	209.0	18.5	63.0	41.0	**100**
200	639.1	408.5	34.0	2856.0	207.5	615.8	416.9	**100**
300	2978.9	2270.0	283.0	16147.0	1948.2	2760.5	2269.2	**100**
400	40317.4	16547.0	897.0	160445.0	8167.0	57017.0	17853.9	**97**
500	45637.3	30480.0	5734.0	165419.0	13164.0	52819.0	28969.2	47
600	88273.7	85589.0	38908.0	140324.0	62248.5	112956.5	77615.3	13
100	**0.4**	**0.4**	**0.2**	**0.7**	**0.3**	**0.4**	**0.4**	**100**
200	**5.8**	**3.5**	**2.1**	**24.0**	**3.0**	**4.9**	**5.1**	**100**
300	**69.2**	**51.7**	**5.0**	242.0	**40.2**	**60.3**	**54.3**	**100**
400	1597.7	471.8	**39.3**	6670.1	349.0	2664.9	721.6	**97**
500	2556.7	1768.2	409.5	6946.8	1039.5	4008.1	1845.4	47
600	8217.1	9612.1	4431.1	10608.0	7021.6	10110.0	7674.2	13

Appendix C

Full Results for the Second Benchmark Test for Monotone MILCP

Here, we present running times, node counts, and optimality gaps for the second test in Section 2.4.5. The tables are split among the different types of feasibility. Note that the instances in Table C.1 have a matrix density of 5 % and 16 % integer variables, instances in Table C.2 have a matrix density of 5 % and 20 % integer variables, and instances in Table C.3 have a matrix density of 5 % and 8 % integer variables.

Table C.1.: Full table of results for the benchmark test with feasibility (a)

		MILCP-PBB			MIQP Reformulation		
Inst.	n	Nodecount	Time	Gap	Nodecount	Time	Gap
0	100	27	3.1159	0.0	68	0.5751	0.0
1	100	15	2.3551	0.0	33	0.1886	0.0
2	100	21	2.6895	0.0	103	0.7351	0.0
3	100	17	2.2424	0.0	15	0.3592	0.0
4	100	15	2.3921	0.0	44	0.1839	0.0
5	100	19	2.2394	0.0	34	0.6797	0.0
6	100	11	2.3276	0.0	18	0.3701	0.0
7	100	15	2.1230	0.0	23	0.4201	0.0
8	100	11	2.1795	0.0	20	0.3384	0.0
9	100	13	1.8813	0.0	14	0.3545	0.0
10	200	35	17.5621	0.0	532	4.0627	0.0
11	200	27	14.9239	0.0	279	3.0247	0.0
12	200	39	18.7982	0.0	399	3.4994	0.0
13	200	43	19.6466	0.0	191	2.5610	0.0

Continued on next page

Table C.1.: Full table of results for the benchmark test with feasibility (a)

		MILCP-PBB			MIQP Reformulation		
Inst.	n	Nodecount	Time	Gap	Nodecount	Time	Gap
14	200	27	16.4442	0.0	127	2.6382	0.0
15	200	35	17.0882	0.0	213	3.1579	0.0
16	200	35	19.3562	0.0	345	3.5871	0.0
17	200	29	18.7861	0.0	110	2.5195	0.0
18	200	37	18.8028	0.0	565	3.8783	0.0
19	200	14	11.0052	0.0	42	2.3146	0.0
20	300	109	99.8787	0.0	7509	189.4373	0.0
21	300	41	52.1754	0.0	2507	51.3995	0.0
22	300	63	71.4824	0.0	2413	51.9769	0.0
23	300	73	71.2369	0.0	2807	54.0534	0.0
24	300	43	64.3237	0.0	1856	31.6593	0.0
25	300	72	78.4768	0.0	1253	22.1835	0.0
26	300	63	69.7968	0.0	1979	40.4144	0.0
27	300	73	78.0971	0.0	2036	39.3973	0.0
28	300	49	61.9105	0.0	2352	47.4422	0.0
29	300	143	123.7982	0.0	2258	56.8950	0.0
30	400	69	155.2925	0.0	3991	164.9751	0.0
31	400	98	186.8513	0.0	16683	638.9243	0.0
32	400	215	354.8989	0.0	8855	348.2817	0.0
33	400	43	134.2780	0.0	1387	62.4837	0.0
34	400	139	240.8656	0.0	12310	426.6363	0.0
35	400	161	244.0987	0.0	12252	442.3353	0.0
36	400	65	166.9921	0.0	2175	87.9427	0.0
37	400	183	235.1562	0.0	16547	431.5883	0.0
38	400	485	462.7339	0.0	10324	414.0996	0.0
39	400	65	152.6845	0.0	1518	63.0660	0.0
40	500	335	608.6910	0.0	265002	10814.2415	0.0
41	500	167	388.2661	0.0	16672	1232.6245	0.0
42	500	117	466.1749	0.0	10873	681.7911	0.0
43	500	131	472.0715	0.0	11418	1039.4614	0.0
44	500	207	279.1084	0.0	5734	409.5402	0.0
45	500	123	154.5519	0.0	14787	1087.2545	0.0
46	500	321	479.0112	0.0	40024	1768.1539	0.0
47	500	134	287.6528	0.0	203450	10816.5233	0.0
48	500	313	390.9182	0.0	165419	6946.8199	0.0
49	500	297	380.2324	0.0	101750	4942.9432	0.0
50	600	293	899.3240	0.0	120917	10824.2120	0.0

Continued on next page

Table C.1.: Full table of results for the benchmark test with feasibility (a)

		MILCP-PBB			MIQP Reformulation		
Inst.	n	Nodecount	Time	Gap	Nodecount	Time	Gap
51	600	327	696.3595	0.0	110087	10829.7680	0.0
52	600	185	512.0450	0.0	38908	4431.0990	0.0
53	600	543	1262.1285	0.0	140118	10829.3220	0.0
54	600	129	794.9032	0.0	156342	10823.6857	0.0
55	600	163	698.5480	0.0	85589	9612.0795	0.0
56	600	201	775.8917	0.0	140324	10608.0138	0.0
57	600	223	736.2904	0.0	138024	10827.5670	0.0
58	600	607	1019.5381	0.0	121674	10827.8138	0.0
59	600	677	1418.2227	0.0	134749	10823.6026	0.0
60	700	249	975.6561	0.0	70611	10836.4064	0.0
61	700	163	1091.6707	0.0	80347	10830.6276	0.0
62	700	303	1174.9706	0.0	54001	10835.4704	0.0
63	700	241	1058.5091	0.0	79913	10838.8376	0.0
64	700	837	1907.9259	0.0	88157	10833.0279	0.0
65	700	1015	2113.7192	0.0	71434	10831.3456	0.0
66	700	301	1187.2362	0.0	73266	10839.0607	0.0
67	700	369	1465.1541	0.0	75375	10841.1106	0.0
68	700	1181	2456.5842	0.0	67206	10829.3712	0.0
69	700	213	1729.2047	0.0	64898	10836.1695	0.0
70	800	209	2894.0162	0.0	50340	10851.0105	0.0
71	800	805	3542.0355	0.0	49637	10853.6512	0.0
72	800	359	1987.7595	0.0	45985	10849.3627	0.0
73	800	1099	4571.4084	0.0	45138	10847.8649	0.0
74	800	126	1556.4838	0.0	48760	10847.7848	0.0
75	800	905	2953.2680	0.0	43308	10846.2972	0.0
76	800	129	1381.9814	0.0	52954	10846.8718	0.0
77	800	246	1621.6197	0.0	64855	10844.3243	0.0
78	800	3069	6670.6305	0.0	47909	10848.3922	0.0
79	800	2215	6693.6248	0.0	51677	10843.3809	0.0
80	900	256	3563.1181	0.0	24062	10865.4706	0.0
81	900	2718	10801.2767	0.0	39963	10861.3130	0.0
82	900	1947	6118.9939	0.0	40548	10857.4068	0.0
83	900	2067	10179.3084	0.0	28530	10862.6133	0.0
84	900	1463	7062.3731	0.0	33852	10861.5166	0.0
85	900	1434	10800.9542	0.0	31292	10871.1387	0.0
86	900	401	4306.1255	0.0	26750	10863.4494	0.0
87	900	1399	10801.3957	0.0	29690	10858.3488	0.0

Continued on next page

Table C.1.: Full table of results for the benchmark test with feasibility (a)

		MILCP-PBB			MIQP Reformulation		
Inst.	n	Nodecount	Time	Gap	Nodecount	Time	Gap
88	900	641	4740.7997	0.0	32362	10857.8563	0.0
89	900	2391	10458.6046	0.0	30177	10844.8206	0.0
90	1000	1163	10802.2212	0.0	21296	10868.1840	0.0
91	1000	2982	10801.7772	0.0	21011	10866.1448	0.0
92	1000	1508	10801.4001	0.0	19055	10859.4321	0.0
93	1000	1335	9664.4435	0.0	20940	10861.2169	0.0
94	1000	830	10802.3529	0.0	19697	10870.9180	0.0
95	1000	2758	10800.8066	0.0	25807	10863.2771	0.0
96	1000	2781	10530.2989	0.0	23170	10864.4643	0.0
97	1000	1283	10801.6599	0.0	25515	10858.4379	0.0
98	1000	1250	10801.9894	0.0	17429	10864.9515	0.0
99	1000	2745	10801.4159	0.0	21093	10854.0908	0.0

Table C.2.: Full table of results for the benchmark test with feasibility (b)

		MILCP-PBB			MIQP Reformulation		
Inst.	n	Nodecount	Time	Gap	Nodecount	Time	Gap
0	100	11	1.8736	0.0	11	0.2401	0.0
1	100	17	2.5071	0.0	35	0.4127	0.0
2	100	11	1.8778	0.0	14	0.3880	0.0
3	100	31	3.0990	0.0	209	0.6444	0.0
4	100	15	2.5030	0.0	51	0.3522	0.0
5	100	15	2.5308	0.0	27	0.3500	0.0
6	100	11	1.7119	0.0	12	0.3445	0.0
7	100	21	2.8848	0.0	72	0.5284	0.0
8	100	9	1.7975	0.0	11	0.2022	0.0
9	100	11	1.9970	0.0	16	0.2678	0.0
10	200	39	20.1349	0.0	478	3.4626	0.0
11	200	75	27.5083	0.0	363	3.3318	0.0
12	200	51	23.1963	0.0	2451	17.3827	0.0
13	200	31	17.3348	0.0	320	2.7709	0.0
14	200	53	24.3716	0.0	435	3.5231	0.0
15	200	21	13.3523	0.0	48	2.0762	0.0
16	200	47	20.3192	0.0	418	3.3012	0.0
17	200	21	14.0000	0.0	34	2.1945	0.0
18	200	57	25.2956	0.0	2856	19.3368	0.0

Continued on next page

Table C.2.: Full table of results for the benchmark test with feasibility (b)

		MILCP-PBB			MIQP Reformulation		
Inst.	n	Nodecount	Time	Gap	Nodecount	Time	Gap
19	200	29	17.1288	0.0	86	2.2520	0.0
20	300	75	85.2808	0.0	2745	57.3262	0.0
21	300	39	50.5451	0.0	341	7.5386	0.0
22	300	149	131.9792	0.0	2090	46.8724	0.0
23	300	35	49.0304	0.0	283	5.0292	0.0
24	300	25	45.2144	0.0	2597	39.8905	0.0
25	300	71	74.3649	0.0	1588	27.9635	0.0
26	300	259	215.7792	0.0	1756	42.2825	0.0
27	300	95	89.7939	0.0	3977	55.6581	0.0
28	300	105	110.0118	0.0	16147	241.9878	0.0
29	300	125	108.0865	0.0	1106	21.9187	0.0
30	400	42	158.0413	0.0	3202	178.1554	0.0
31	400	255	335.9703	0.0	32060	1002.8782	0.0
32	400	309	361.5803	0.0	34792	1298.5175	0.0
33	400	75	206.8250	0.0	10523	471.8054	0.0
34	400	227	364.7588	0.0	8167	433.7521	0.0
35	400	92	210.3800	0.0	62434	2256.8009	0.0
36	400	37	138.3447	0.0	897	39.3350	0.0
37	400	243	310.8748	0.0	57017	2005.0877	0.0
38	400	165	287.8678	0.0	7969	349.0208	0.0
39	400	357	395.5973	0.0	267946	7095.4798	0.0
40	500	113	385.2338	0.0	13164	806.6624	0.0
41	500	511	845.0595	0.0	30480	2447.1081	0.0
42	500	769	1563.5448	0.0	204209	10813.9150	0.0
43	500	1009	1436.5816	0.0	171660	10815.6210	0.0
44	500	685	881.8793	0.0	48103	4008.1435	0.0
45	500	2393	3237.3454	0.0	157972	10816.5673	0.0
46	500	385	421.6556	0.0	52819	3227.4129	0.0
47	500	485	746.2523	0.0	82042	4638.5376	0.0
48	500	543	532.2371	0.0	187584	10814.5764	0.0
49	500	155	227.8922	0.0	34045	2399.3100	0.0
50	600	1221	2966.6930	0.0	115289	10824.2649	0.0
51	600	539	1488.9798	0.0	80734	10823.4649	0.0
52	600	377	781.4358	0.0	120840	10824.6150	0.0
53	600	473	1174.9950	0.0	120541	10824.3100	0.0
54	600	411	1146.9656	0.0	123993	10823.1460	0.0
55	600	531	1376.3102	0.0	135998	10823.9850	0.0

Continued on next page

Table C.2.: Full table of results for the benchmark test with feasibility (b)

		MILCP-PBB			MIQP Reformulation		
Inst.	n	Nodecount	Time	Gap	Nodecount	Time	Gap
56	600	443	862.3373	0.0	108494	10822.6135	0.0
57	600	347	708.2811	0.0	134048	10824.2348	0.0
58	600	143	722.6545	0.0	15769	1731.9034	0.0
59	600	355	1161.8243	0.0	125770	10824.3285	0.0
60	700	883	2755.3962	0.0	86915	10832.8358	0.0
61	700	981	3975.7980	0.0	60690	10834.3674	0.0
62	700	165	1288.9324	0.0	59965	10834.6514	0.0
63	700	383	1461.5977	0.0	66430	10835.0290	0.0
64	700	1031	2620.0381	0.0	71492	10837.0440	0.0
65	700	3183	5449.0948	0.0	77091	10834.8594	0.0
66	700	1219	4465.0251	0.0	64723	10832.1042	0.0
67	700	308	1820.4327	0.0	68971	10830.7743	0.0
68	700	801	2834.0616	0.0	57584	10831.4539	0.0
69	700	494	2710.5223	0.0	64278	10833.1984	0.0
70	800	1367	4188.1333	0.0	48056	10844.6034	0.0
71	800	1394	6301.3453	0.0	35039	10843.7623	0.0
72	800	2682	10800.9745	0.0	38116	10846.3990	0.0
73	800	319	2445.0687	0.0	54200	10847.9241	0.0
74	800	3698	10801.9776	0.0	41462	10850.7178	0.0
75	800	1635	4952.7012	0.0	37047	10850.4457	0.0
76	800	206	2589.2310	0.0	42891	10847.8842	0.0
77	800	1565	7296.9954	0.0	32957	10845.7794	0.0
78	800	5283	10801.1113	0.0	42228	10841.0241	0.0
79	800	1399	5201.9637	0.0	56617	10843.3123	0.0
80	900	1068	10803.4404	0.0	26577	10856.2667	0.0
81	900	1539	10800.0682	0.0	23554	10853.1402	0.0
82	900	3585	10801.5707	0.0	30402	10866.2853	0.0
83	900	2131	10801.4520	0.0	31559	10857.8554	0.0
84	900	996	10801.3616	0.0	26331	10860.5435	0.0
85	900	1157	10803.2434	0.0	1	192.6932	inf
86	900	2086	10801.6023	0.0	22924	10859.6405	0.0
87	900	3121	10800.1144	0.0	27014	10855.2115	0.0
88	900	2281	10755.1982	0.0	32147	10855.9983	0.0
89	900	1440	8768.8400	0.0	29748	10841.0772	0.0
90	1000	1267	10686.1705	0.0	18266	10865.9510	0.0
91	1000	505	7181.2388	0.0	20744	10864.9113	0.0
92	1000	387	5606.9840	0.0	18491	10863.1418	0.0

Continued on next page

Table C.2.: Full table of results for the benchmark test with feasibility (b)

		MILCP-PBB			MIQP Reformulation		
Inst.	n	Nodecount	Time	Gap	Nodecount	Time	Gap
93	1000	1073	10800.9104	0.0	18227	10859.5245	0.0
94	1000	1420	10802.2677	0.0	18885	10852.9994	0.0
95	1000	1877	10800.4012	0.0	16198	10853.8927	0.0
96	1000	244	6376.3823	0.0	18063	10858.1245	0.0
97	1000	50	5808.5997	0.0	20066	10862.6262	0.0
98	1000	1419	10800.7780	0.0	23980	10858.5239	0.0
99	1000	1506	10800.1857	0.0	20533	10859.9718	0.0

Table C.3.: Full table of results for the benchmark test with feasibility (c)

		MILCP-PBB			MIQP Reformulation		
Inst.	n	Nodecount	Time	Gap	Nodecount	Time	Gap
0	100	17	2.2032	0.0	69	0.3876	0.0
1	100	17	2.2020	0.0	35	0.3816	0.0
2	100	23	2.7185	0.0	46	0.3951	0.0
3	100	17	2.4266	0.0	25	0.3482	0.0
4	100	21	2.5839	0.0	73	0.1972	0.0
5	100	17	2.3956	0.0	60	0.1806	0.0
6	100	17	2.3847	0.0	31	0.2105	0.0
7	100	25	2.7450	0.0	93	0.3988	0.0
8	100	21	2.5308	0.0	26	0.3163	0.0
9	100	23	2.6372	0.0	64	0.3479	0.0
10	200	29	14.2188	0.0	142	3.0308	0.0
11	200	33	18.1685	0.0	768	6.0798	0.0
12	200	47	25.6820	0.0	873	7.5518	0.0
13	200	49	26.3865	0.0	451	4.4537	0.0
14	200	53	28.1232	0.0	1041	13.0250	0.0
15	200	71	33.9903	0.0	1057	8.7275	0.0
16	200	47	22.6940	0.0	220	3.0421	0.0
17	200	33	18.3735	0.5	183	2.9801	0.0
18	200	41	21.0298	0.0	444	4.0270	0.0
19	200	107	52.9182	0.0	2650	23.9550	0.0
20	300	123	135.1874	0.0	2729	87.4795	0.0
21	300	75	109.0742	0.0	2432	53.4491	0.0
22	300	171	160.6897	0.0	4016	148.4997	0.0
23	300	103	122.9437	0.0	2876	119.3457	0.0

Continued on next page

Table C.3.: Full table of results for the benchmark test with feasibility (c)

		MILCP-PBB			MIQP Reformulation		
Inst.	n	Nodecount	Time	Gap	Nodecount	Time	Gap
24	300	125	137.8550	0.0	1090	30.7861	0.0
25	300	77	97.9103	0.0	1993	56.0017	0.0
26	300	79	86.1865	0.0	2139	50.6660	0.0
27	300	203	168.5056	0.0	5906	203.0584	0.0
28	300	69	66.8797	0.0	2282	69.0258	0.0
29	300	71	95.9877	0.0	2197	50.7216	0.0
30	400	779	847.8113	0.0	121573	4819.6968	0.0
31	400	339	519.3352	0.0	51856	3256.8336	0.0
32	400	519	779.5839	0.0	107926	3578.9797	0.0
33	400	171	341.3093	0.0	62027	3165.3430	0.0
34	400	267	444.8971	0.0	245500	10808.8307	0.1
35	400	615	955.3639	0.0	70852	2664.9435	0.0
36	400	261	379.4113	0.0	160445	6670.0696	0.0
37	400	165	278.4771	0.0	29917	1243.3834	0.0
38	400	163	343.1481	0.0	47667	1712.7171	0.0
39	400	423	651.1013	0.0	134887	4788.1721	0.0
40	500	1059	2464.7738	0.0	163745	10815.2826	0.2
41	500	2059	4223.0813	0.0	165779	10818.6912	0.2
42	500	763	1678.4753	0.0	140830	10817.5013	0.2
43	500	285	765.3047	0.0	198463	10817.2753	0.1
44	500	923	885.9161	0.0	136841	10815.9276	0.2
45	500	407	647.9789	0.0	126879	10816.3697	0.1
46	500	265	284.3640	0.0	128834	10816.3495	0.3
47	500	665	1058.7074	0.0	140357	10815.7690	0.1
48	500	485	571.5811	0.0	174696	10816.4255	0.2
49	500	629	706.5074	0.0	189088	10815.0503	0.3
50	600	10808	10800.2888	0.9	79144	10826.8733	0.5
51	600	3943	3818.3468	0.0	100441	10828.4613	0.5
52	600	683	1024.9046	0.0	107480	10828.8295	0.4
53	600	631	2597.7218	0.0	97200	10826.0636	0.3
54	600	413	1049.1375	0.0	94083	10825.6469	0.5
55	600	653	2568.5341	0.0	104809	10828.5226	0.4
56	600	54	609.9500	1.0	97417	10826.1439	0.4
57	600	1453	2971.0164	0.0	102075	10829.7080	0.5
58	600	9	232.5856	1.0	74238	10828.7163	0.5
59	600	678	1588.2402	0.8	91260	10830.4672	0.5
60	700	249	1128.5292	1.0	41533	10849.1465	0.6

Continued on next page

Table C.3.: Full table of results for the benchmark test with feasibility (c)

		MILCP-PBB			MIQP Reformulation		
Inst.	n	Nodecount	Time	Gap	Nodecount	Time	Gap
61	700	273	1297.1256	0.9	57176	10837.0305	0.6
62	700	4066	10801.5454	0.7	60318	10835.0205	0.6
63	700	13	444.0867	1.0	50058	10841.8503	0.4
64	700	805	1967.7534	1.0	45885	10844.9726	0.6
65	700	3	423.5964	1.0	58474	10842.7989	0.6
66	700	8193	10800.4775	1.0	46798	10842.8968	0.7
67	700	449	1834.5219	1.0	56457	10840.2731	0.6
68	700	2154	10801.2486	1.0	59715	10843.3221	0.6
69	700	815	4528.8139	0.0	58624	10847.4065	0.6
70	800	8	1325.5795	1.0	37286	10858.9890	0.6
71	800	1412	10802.3115	1.0	32379	10865.9849	0.6
72	800	1263	9918.1721	0.0	33698	10861.6950	0.6
73	800	283	2182.9504	1.0	26555	10859.3335	0.7
74	800	2787	10800.9629	0.9	32880	10860.2008	0.7
75	800	5184	10801.2334	1.0	33831	10860.9934	0.7
76	800	2973	10800.1716	1.0	33833	10853.8966	0.7
77	800	711	4456.3225	1.0	24822	10862.4918	0.7
78	800	29	1585.9883	1.0	29949	10863.7911	0.7
79	800	5	1418.6417	1.0	1	234.0652	inf
80	900	346	4289.8207	1.0	22578	10869.1079	0.7
81	900	1004	10802.9810	1.0	21926	10859.6112	0.7
82	900	1527	10800.4423	1.0	17300	10863.7038	0.8
83	900	539	3108.7699	1.0	20772	10861.9778	0.7
84	900	921	10803.1083	0.9	19133	10856.5056	0.7
85	900	1443	10801.9199	1.0	19178	10859.8103	0.7
86	900	135	3343.6416	1.0	17451	10858.2922	0.8
87	900	1154	10802.4067	1.0	26547	10849.0477	0.7
88	900	80	2913.2712	1.0	20462	10845.4186	0.7
89	900	1	1271.1297	1.0	19289	10844.5248	0.7
90	1000	163	5233.4671	1.0	14795	10859.0219	0.8
91	1000	2678	10800.5171	1.0	17441	10856.4719	0.8
92	1000	73	4245.0406	1.0	15919	10858.5450	0.8
93	1000	862	10802.7972	1.0	16530	10871.7120	0.8
94	1000	1326	10801.8961	1.0	18380	10868.6293	0.8
95	1000	76	4242.0261	1.0	17985	10866.4091	0.8
96	1000	1	2617.2543	1.0	22577	10864.8447	0.8
97	1000	1045	10801.3138	1.0	19917	10861.5392	0.8

Continued on next page

Table C.3.: Full table of results for the benchmark test with feasibility (c)

		MILCP-PBB			MIQP Reformulation		
Inst.	n	Nodecount	Time	Gap	Nodecount	Time	Gap
98	1000	68	3136.0200	1.0	24022	10850.9295	0.8
99	1000	30	3088.7299	1.0	19310	10854.8429	0.8

Appendix D

Detailed Description of the Test Set for Non-Monotone MILCP

The parameters for MATLAB's sprandsym function were the following: matrix sizes were set to

$$n \in \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\},$$

while the range of the matrices was set to 100 and the matrix density was set to 10 %. The set of integer variables I has been chosen as a random sample containing 10 % of the total number of variables.

As described in Appendix A, we built vectors $q \in \mathbb{R}^n$ for the four different "degrees of feasibility"; see Section 4.4. 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)).

Appendix E

Tables of Aggregated Results of Other Settings for Non-Monotone MILCP

In what follows, we include all tables for the aggregated running times and node counts of the settings not reported in Section 4.4.

E.1. Branching Rule Test

Table E.1.: Aggregated nodecounts (top) and runtimes (bottom) for the branching rule test with random choice

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
10	5.8	**5.0**	**1.0**	29.0	**3.0**	**7.0**	5.7	**100**
20	53.6	28.0	7.0	367.0	15.0	66.0	44.4	**100**
30	1144.8	413.0	**25.0**	7937.0	55.0	1443.0	495.4	**100**
40	14319.9	753.0	53.0	84929.0	67.5	6732.0	1428.2	**100**
50	1707.9	95.0	81.0	10699.0	89.0	451.0	334.5	68
60	99.0	99.0	99.0	99.0	99.0	99.0	99.0	30
10	**0.0**	**0.0**	**0.0**	**0.1**	**0.0**	**0.0**	**0.0**	**100**
20	**0.2**	**0.1**	**0.0**	1.4	**0.1**	**0.2**	**0.2**	**100**
30	9.9	3.6	0.2	68.0	**0.5**	12.4	6.8	**100**
40	296.1	12.4	0.9	1848.3	1.2	112.4	41.2	**100**
50	50.6	2.8	2.5	318.6	2.7	12.4	13.2	68
60	4.8	4.8	4.8	4.8	4.8	4.8	4.8	30

Table E.2.: Aggregated nodecounts (top) and runtimes (bottom) for the branching rule test with most fractional branching and integer branching done first

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
10	5.7	**5.0**	**1.0**	29.0	**3.0**	**7.0**	5.6	**100**
20	44.9	**18.0**	**5.0**	289.0	12.5	**52.0**	36.6	**100**
30	**365.4**	185.0	33.0	**3019.0**	57.0	501.0	**223.6**	**100**
40	1765.9	321.0	41.0	10215.0	137.5	687.0	535.5	**100**
50	235.0	159.0	73.0	761.0	86.0	240.0	185.1	**100**
60	107.0	107.0	107.0	107.0	107.0	107.0	107.0	78
10	**0.0**	**0.0**	**0.0**	**0.1**	**0.0**	**0.0**	**0.0**	**100**
20	**0.2**	**0.1**	**0.0**	1.2	**0.1**	**0.2**	**0.2**	**100**
30	**3.3**	1.7	0.3	**27.5**	**0.5**	4.4	**2.8**	**100**
40	31.1	5.4	0.8	182.1	2.5	11.7	13.3	**100**
50	6.8	4.7	2.4	21.0	2.7	7.1	6.0	**100**
60	5.1	5.1	5.1	5.1	5.1	5.1	5.1	78

Table E.3.: Aggregated nodecounts (top) and runtimes (bottom) for the branching rule test with pseudocost branching

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
10	**5.5**	**5.0**	**1.0**	**13.0**	**3.0**	**7.0**	**5.4**	**100**
20	**39.2**	24.0	7.0	**137.0**	17.0	**52.0**	**35.4**	**100**
30	425.8	197.0	31.0	3541.0	51.0	491.0	242.5	**100**
40	**826.9**	215.0	63.0	**8045.0**	76.0	**621.5**	**332.3**	**100**
50	158.7	89.0	79.0	575.0	86.0	98.0	126.9	**100**
60	109.0	109.0	109.0	109.0	109.0	109.0	109.0	80
10	**0.0**	**0.0**	**0.0**	**0.1**	**0.0**	**0.0**	**0.0**	**100**
20	**0.2**	**0.1**	**0.0**	**0.6**	**0.1**	**0.2**	**0.2**	**100**
30	4.6	2.1	0.3	40.4	**0.5**	5.2	3.6	**100**
40	**17.4**	4.5	1.3	**169.2**	1.6	12.8	**9.0**	**100**
50	5.9	3.3	2.9	21.4	3.2	3.6	5.0	**100**
60	6.1	6.1	6.1	6.1	6.1	6.1	6.1	80

Table E.4.: Aggregated nodecounts (top) and runtimes (bottom) for the branching rule test with preprocessed order branching

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
10	6.8	**5.0**	**1.0**	21.0	**3.0**	11.0	6.7	**100**
20	58.6	34.0	9.0	317.0	20.5	64.5	49.6	**100**
30	1581.1	701.0	45.0	9911.0	105.0	1659.0	653.9	**100**
40	22282.3	5235.0	63.0	138263.0	133.0	30240.5	3105.7	88
50	11068.4	101.0	67.0	75951.0	87.0	593.0	479.5	38
60	165.0	165.0	165.0	165.0	165.0	165.0	165.0	28
10	**0.0**	**0.0**	**0.0**	**0.1**	**0.0**	**0.0**	**0.0**	**100**
20	0.3	0.2	0.1	1.3	**0.1**	0.3	0.3	**100**
30	14.1	6.1	0.6	91.5	1.1	14.4	8.9	**100**
40	480.8	84.2	1.4	3507.4	2.6	564.0	84.2	88
50	360.2	3.6	2.7	2473.5	3.4	17.3	23.1	38
60	8.4	8.4	8.4	8.4	8.4	8.4	8.4	28

E.2. Node Selection Test

Table E.5.: Aggregated nodecounts (top) and runtimes (bottom) for the node selection test with breadth-first search

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
10	6.6	**4.0**	**1.0**	39.0	**3.0**	**7.0**	6.4	**100**
20	77.5	54.0	**5.0**	293.0	27.0	100.5	65.6	**100**
30	727.7	535.0	89.0	4221.0	323.0	837.0	551.0	**100**
40	5528.4	2878.0	85.0	19737.0	853.5	8993.0	2457.9	**100**
50	6144.7	3585.0	795.0	21983.0	2154.0	6171.0	3736.7	82
60	16219.0	16219.0	16219.0	16219.0	16219.0	16219.0	16219.0	35
10	**0.0**	**0.0**	**0.0**	**0.1**	**0.0**	**0.0**	**0.0**	**100**
20	0.3	0.2	0.0	1.2	0.1	0.3	0.3	**100**
30	6.3	4.7	0.9	35.3	3.2	8.1	5.6	**100**
40	96.8	52.4	1.7	350.8	13.6	150.3	50.7	**100**
50	189.0	105.9	24.1	711.6	64.7	175.9	115.4	82
60	788.4	788.4	788.4	788.4	788.4	788.4	788.4	35

Table E.6.: Aggregated nodecounts (top) and runtimes (bottom) for the node selection test with lower bound push

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
10	6.2	5.0	**1.0**	33.0	**3.0**	**7.0**	6.0	**100**
20	**44.8**	23.0	**5.0**	**283.0**	13.0	**38.0**	**36.6**	**100**
30	**331.1**	155.0	45.0	**2465.0**	87.0	**409.0**	**225.6**	**100**
40	**1224.3**	272.0	35.0	**7097.0**	113.0	990.0	462.6	**100**
50	719.6	585.0	119.0	1795.0	311.0	958.0	545.1	**100**
60	1913.0	1913.0	1913.0	1913.0	1913.0	1913.0	1913.0	**92**
10	**0.0**	**0.0**	**0.0**	**0.1**	**0.0**	**0.0**	**0.0**	**100**
20	**0.2**	**0.1**	**0.0**	**1.1**	0.1	**0.1**	**0.2**	**100**
30	**2.8**	**1.4**	0.4	**20.7**	0.8	3.7	**2.5**	**100**
40	**21.0**	4.9	0.7	**114.1**	1.9	17.3	11.0	**100**
50	17.7	12.5	3.9	48.2	8.1	21.4	14.8	**100**
60	92.9	92.9	92.9	92.9	92.9	92.9	92.9	**92**

E.3. Warmstart Test

Table E.7.: Aggregated nodecounts (top) and runtimes (bottom) for the warm start test with warmstart off

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
10	6.0	5.0	**1.0**	31.0	**3.0**	7.0	5.9	**100**
20	49.2	18.0	5.0	345.0	11.0	56.5	38.6	**100**
30	431.1	149.0	33.0	3545.0	49.0	429.0	227.4	**100**
40	1726.7	207.0	33.0	10255.0	64.0	651.0	420.0	**100**
50	141.0	75.0	65.0	**517.0**	75.0	90.0	**112.7**	**100**
60	87.0	87.0	87.0	87.0	87.0	87.0	87.0	85
10	**0.0**	**0.0**	**0.0**	**0.1**	**0.0**	**0.0**	**0.0**	**100**
20	0.2	**0.1**	**0.0**	1.4	0.1	0.2	0.2	**100**
30	4.0	1.4	0.3	33.5	0.5	3.9	3.0	**100**
40	31.5	3.5	0.7	189.6	1.2	**11.7**	11.9	**100**
50	**4.1**	2.5	2.2	**13.6**	2.4	2.8	**3.7**	**100**
60	4.7	4.7	4.7	4.7	4.7	4.7	4.7	85

Table E.8.: Aggregated nodecounts (top) and runtimes (bottom) for the warm start test using PStart/DStart

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
10	5.4	**4.0**	**1.0**	**19.0**	**3.0**	7.0	5.3	**100**
20	34.0	20.0	5.0	177.0	**9.0**	34.0	29.2	**100**
30	**268.0**	**125.0**	**27.0**	**1473.0**	47.0	367.0	178.5	**100**
40	**879.8**	285.0	27.0	6391.0	61.5	914.5	350.7	**100**
50	**147.9**	81.0	63.0	581.0	67.0	88.0	112.8	**100**
60	89.0	89.0	89.0	89.0	89.0	89.0	89.0	**90**
10	**0.0**	**0.0**	**0.0**	**0.1**	**0.0**	**0.0**	**0.0**	**100**
20	0.2	**0.1**	**0.0**	0.8	**0.0**	0.2	0.2	**100**
30	**2.6**	**1.2**	0.3	**13.5**	0.5	3.5	**2.3**	**100**
40	**17.1**	5.0	**0.6**	126.2	1.2	17.1	9.2	**100**
50	4.5	2.8	2.1	16.2	2.3	2.9	3.9	**100**
60	4.7	4.7	4.7	4.7	4.7	4.7	4.7	**90**

E.4. Valid Inequalities Test

Table E.9.: Aggregated nodecounts (top) and runtimes (bottom) for the valid inequality test with all binary cuts

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
10	6.0	**5.0**	**1.0**	31.0	**3.0**	**7.0**	5.9	**100**
20	49.2	**18.0**	**5.0**	345.0	**11.0**	**56.5**	**38.6**	**100**
30	431.2	**149.0**	**33.0**	3545.0	**49.0**	437.0	**227.4**	**100**
40	1719.7	**207.0**	33.0	**10255.0**	**64.0**	**620.0**	**414.0**	**100**
50	137.9	**75.0**	**65.0**	495.0	**75.0**	**90.0**	111.6	**100**
60	87.0	87.0	87.0	87.0	87.0	87.0	87.0	**85**
10	**0.0**	**0.0**	**0.0**	**0.1**	**0.0**	**0.1**	**0.0**	**100**
20	**0.2**	**0.1**	**0.0**	**1.4**	**0.1**	**0.2**	**0.2**	**100**
30	4.1	1.6	**0.3**	33.8	**0.5**	**4.0**	3.2	**100**
40	32.0	3.7	**0.6**	195.4	**1.3**	**11.0**	**11.9**	**100**
50	4.2	2.7	**2.1**	**13.5**	2.5	3.0	3.8	**100**
60	4.8	4.8	4.8	4.8	4.8	4.8	4.8	**85**

Table E.10.: Aggregated nodecounts (top) and runtimes (bottom) for the valid inequality test with all complementarity cuts

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
10	**5.7**	**5.0**	**1.0**	**27.0**	**3.0**	**7.0**	**5.6**	**100**
20	**46.8**	20.0	**5.0**	**219.0**	12.5	59.0	**38.6**	**100**
30	**404.9**	185.0	37.0	**2559.0**	55.0	547.0	238.0	**100**
40	1599.8	227.0	**25.0**	10439.0	67.5	757.0	429.7	98
50	131.3	83.0	71.0	429.0	76.0	92.0	111.4	88
60	**79.0**	**79.0**	**79.0**	**79.0**	**79.0**	**79.0**	**79.0**	58
10	0.1	**0.0**	**0.0**	**0.1**	**0.0**	**0.1**	0.1	**100**
20	1.0	0.3	**0.0**	5.4	**0.1**	1.3	0.9	**100**
30	35.3	12.3	2.1	263.9	4.3	44.1	19.6	**100**
40	295.6	48.9	1.9	1963.2	10.6	144.9	71.0	98
50	52.9	24.8	19.4	222.4	21.4	30.3	35.1	88
60	33.9	33.9	33.9	33.9	33.9	33.9	33.9	58

Table E.11.: Aggregated nodecounts (top) and runtimes (bottom) for the valid inequality test with all simple cuts

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
10	**5.7**	**5.0**	**1.0**	**27.0**	**3.0**	**7.0**	**5.6**	**100**
20	**46.8**	20.0	**5.0**	**219.0**	12.5	59.0	**38.6**	**100**
30	405.6	185.0	37.0	**2559.0**	55.0	547.0	238.4	**100**
40	**1599.6**	225.0	**25.0**	10439.0	67.5	757.0	429.3	98
50	**129.9**	83.0	71.0	**419.0**	76.0	92.0	**110.8**	85
60	**79.0**	**79.0**	**79.0**	**79.0**	**79.0**	**79.0**	**79.0**	58
10	0.1	0.1	**0.0**	**0.1**	**0.0**	**0.1**	0.1	**100**
20	0.9	0.3	**0.0**	5.2	**0.1**	1.2	0.9	**100**
30	35.6	12.9	2.1	264.5	4.1	46.8	19.7	**100**
40	294.1	49.5	2.0	1956.1	10.9	142.7	71.4	98
50	52.4	25.4	18.5	217.3	21.9	31.0	35.2	85
60	34.4	34.4	34.4	34.4	34.4	34.4	34.4	58

E.5. Benchmark Test

Table E.12.: Aggregated nodecounts (top) and runtimes (bottom) for the benchmark test for the MILP reformulation

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
10	284.1	3.0	**1.0**	5109.0	**2.5**	**6.0**	34.0	**100**
20	698.2	10.0	**1.0**	5120.0	**6.0**	**18.2**	99.6	**100**
30	9470.2	5142.0	**12.0**	92191.0	**28.0**	6664.0	2146.0	82
40	499557.1	85193.5	**28.0**	4061416.0	14270.5	365148.8	55051.0	50
50	3571474.1	115450.0	5169.0	24167523.0	93313.0	262775.5	179955.5	45
60	6616978.0	6616978.0	6616978.0	6616978.0	6616978.0	6616978.0	6616978.0	8
10	**0.1**	**0.1**	0.1	0.2	**0.1**	**0.1**	**0.1**	**100**
20	**0.0**	**0.0**	**0.0**	**0.2**	**0.0**	**0.0**	**0.0**	**100**
30	**0.4**	**0.2**	**0.0**	**3.0**	**0.0**	**0.3**	**0.4**	82
40	38.4	3.6	**0.1**	467.6	**0.9**	17.0	**11.3**	50
50	213.0	6.6	**0.6**	1440.7	5.9	15.6	22.5	45
60	476.7	476.7	476.7	476.7	476.7	476.7	476.7	8

Appendix F

Full Results for the Benchmark Test for Non-Monotone MILCP

Here, we present running times, node counts, and optimality gaps for the benchmark test in Section 4.4.5.

Table F.1.: Full table of results for the benchmark test

		MILCP-PBB			MILP Reformulation		
Inst.	n	Nodecount	Time	Gap	Nodecount	Time	Gap
0	10	5	0.0485	0.0	3	0.0774	0.0
1	10	5	0.0674	0.0	1	0.0716	0.0
2	10	3	0.0645	0.0	3	0.0813	0.0
3	10	3	0.0661	0.0	3	0.0855	0.0
4	10	5	0.0645	0.0	4	0.0766	0.0
5	10	5	0.0651	0.0	3	0.0726	0.0
6	10	5	0.0678	0.0	3	0.0730	0.0
7	10	1	0.0602	0.0	3	0.0744	0.0
8	10	3	0.0592	0.0	3	0.0744	0.0
9	10	1	0.0589	0.0	1	0.0754	0.0
10	10	7	0.0672	0.0	9	0.0750	0.0
11	10	3	0.0575	0.0	5	0.0743	0.0
12	10	3	0.0617	0.0	4	0.0812	0.0
13	10	3	0.0592	0.0	1	0.0789	0.0
14	10	9	0.0656	0.0	6	0.0699	0.0
15	10	7	0.0646	0.0	4	0.0738	0.0
16	10	5	0.0589	0.0	1	0.0502	0.0

Continued on next page

Table F.1.: Full table of results for the benchmark test

		MILCP-PBB			MILP Reformulation		
Inst.	n	Nodecount	Time	Gap	Nodecount	Time	Gap
17	10	3	0.0578	0.0	1	0.0717	0.0
18	10	19	0.0706	0.0	15	0.0800	0.0
19	10	7	0.0642	0.0	5109	0.1798	0.0
20	10	3	0.0584	0.0	3	0.0704	0.0
21	10	3	0.0592	0.0	1	0.0723	0.0
22	10	11	0.0707	0.0	1004	0.0911	0.0
23	10	9	0.0600	0.0	6	0.0978	0.0
24	10	5	0.0617	0.0	3	0.0700	0.0
25	10	7	0.0637	0.0	4	0.0699	0.0
26	10	13	0.0731	0.0	6	0.0821	0.0
27	10	31	0.0783	0.0	10	0.0713	0.0
28	10	1	0.0545	0.0	1	0.0796	0.0
29	10	7	0.0617	0.0	1	0.0719	0.0
30	10	1	0.0562	0.0	3	0.0835	0.0
31	10	9	0.0621	0.0	7	0.0923	0.0
32	10	3	0.0547	0.0	3	0.0696	0.0
33	10	11	0.0691	0.0	5109	0.1726	0.0
34	10	3	0.0561	0.0	5	0.0698	0.0
35	10	7	0.0598	0.0	6	0.0740	0.0
36	10	1	0.0531	0.0	1	0.0663	0.0
37	10	3	0.0561	0.0	4	0.0890	0.0
38	10	1	0.0548	0.0	1	0.0670	0.0
39	10	9	0.0658	0.0	5	0.0767	0.0
40	20	15	0.1032	0.0	5117	0.2371	0.0
41	20	11	0.0983	0.0	10	0.0826	0.0
42	20	29	0.1577	0.0	5120	0.2500	0.0
43	20	13	0.0885	0.0	10	0.0904	0.0
44	20	15	0.0639	0.0	7	0.0173	0.0
45	20	5	0.0158	0.0	6	0.0330	0.0
46	20	7	0.0523	0.0	7	0.0238	0.0
47	20	15	0.0502	0.0	9	0.0242	0.0
48	20	11	0.0674	0.0	15	0.0309	0.0
49	20	25	0.0993	0.0	10	0.0287	0.0
50	20	5	0.0314	0.0	1	0.0246	0.0
51	20	13	0.0625	0.0	5	0.0217	0.0
52	20	11	0.0652	0.0	8	0.0247	0.0
53	20	7	0.0485	0.0	3	0.0223	0.0

Continued on next page

Table F.1.: Full table of results for the benchmark test

		MILCP-PBB			MILP Reformulation		
Inst.	n	Nodecount	Time	Gap	Nodecount	Time	Gap
54	20	87	0.3608	0.0	6	0.0305	0.0
55	20	35	0.1411	0.0	18	0.0322	0.0
56	20	25	0.1146	0.0	5	0.0219	0.0
57	20	137	0.5641	0.0	1015	0.0532	0.0
58	20	13	0.0695	0.0	1004	0.0378	0.0
59	20	19	0.0922	0.0	12	0.0287	0.0
60	20	21	0.0974	0.0	5111	0.1629	0.0
61	20	345	1.3971	0.0	23	0.0248	0.0
62	20	17	0.0797	0.0	8	0.0176	0.0
63	20	19	0.0907	0.0	6	0.0114	0.0
64	20	277	1.1314	0.0	5115	0.1556	0.0
65	20	9	0.0384	0.0	15	0.0226	0.0
66	20	13	0.0602	0.0	6	0.0164	0.0
67	20	61	0.2377	0.0	8	0.0180	0.0
68	20	9	0.0410	0.0	8	0.0161	0.0
69	20	129	0.4705	0.0	15	0.0133	0.0
70	20	47	0.1955	0.0	5120	0.1768	0.0
71	20	15	0.0668	0.0	8	0.0117	0.0
72	20	77	0.3048	0.0	3	0.0156	0.0
73	20	55	0.2201	0.0	10	0.0119	0.0
74	20	23	0.0962	0.0	19	0.0189	0.0
75	20	11	0.0499	0.0	23	0.0184	0.0
76	20	79	0.3182	0.0	6	0.0133	0.0
77	20	9	0.0403	0.0	14	0.0105	0.0
78	20	169	0.6690	0.0	17	0.0155	0.0
79	20	85	0.3371	0.0	4	0.0150	0.0
80	30	49	0.4500	0.0	1510	0.0722	0.0
81	30	37	0.3598	0.0	6642	0.3085	0.0
82	30	39	0.3653	0.0	6793947	3602.7226	inf
83	30	47	0.4403	0.0	5130	0.2438	0.0
84	30	39	0.3721	0.0	21	0.0346	0.0
85	30	35	0.3382	0.0	18	0.0328	0.0
86	30	45	0.4277	0.0	25	0.0429	0.0
87	30	55	0.5133	0.0	9677	0.4744	0.0
88	30	35	0.3390	0.0	25	0.0389	0.0
89	30	43	0.4034	0.0	28	0.0396	0.0
90	30	151	1.5006	0.0	7177326	3603.9318	1.0

Continued on next page

Table F.1.: Full table of results for the benchmark test

		MILCP-PBB			MILP Reformulation		
Inst.	n	Nodecount	Time	Gap	Nodecount	Time	Gap
91	30	109	1.0723	0.0	6659	0.3106	0.0
92	30	717	7.1047	0.0	5162	0.2755	0.0
93	30	755	7.5426	0.0	9014631	3601.4398	1.0
94	30	1009	9.5735	0.0	6755328	3605.3493	1.0
95	30	187	1.8717	0.0	5134	0.2333	0.0
96	30	63	0.5485	0.0	92191	2.9865	0.0
97	30	401	3.6816	0.0	8960150	3602.8095	0.1
98	30	3545	33.4889	0.0	8136	0.2947	0.0
99	30	99	0.8690	0.0	5127	0.2208	0.0
100	30	2611	22.8850	0.0	6664	0.3054	0.0
101	30	365	3.4610	0.0	5150	0.2645	0.0
102	30	807	7.0044	0.0	5144	0.2351	0.0
103	30	519	4.8498	0.0	7540	0.3528	0.0
104	30	107	0.8586	0.0	20	0.0356	0.0
105	30	311	2.8744	0.0	5166	0.3012	0.0
106	30	241	2.1818	0.0	12	0.0301	0.0
107	30	115	1.1005	0.0	9336339	3602.6633	inf
108	30	33	0.3283	0.0	6670	0.3510	0.0
109	30	429	3.4316	0.0	38	0.0370	0.0
110	30	149	1.4363	0.0	5142	0.2255	0.0
111	30	365	3.6856	0.0	5112	0.2279	0.0
112	30	525	4.9961	0.0	13	0.0287	0.0
113	30	333	3.2030	0.0	32212	1.1470	0.0
114	30	501	4.9082	0.0	57703	2.1175	0.0
115	30	83	0.7411	0.0	12	0.0277	0.0
116	30	107	0.9802	0.0	20179	0.8147	0.0
117	30	1219	12.3243	0.0	5112	0.2412	0.0
118	30	415	3.9041	0.0	5143	0.2909	0.0
119	30	99	0.9028	0.0	6493392	3604.2579	1.0
120	40	33	0.6236	0.0	5111	0.3589	0.0
121	40	73	1.3336	0.0	10663573	3600.3667	1.0
122	40	53	1.0124	0.0	59333	2.7933	0.0
123	40	67	1.2538	0.0	1998062	88.4977	0.0
124	40	65	1.2418	0.0	12579931	3601.1637	1.0
125	40	53	1.1006	0.0	28	0.0613	0.0
126	40	73	1.3673	0.0	58316	2.8901	0.0
127	40	67	1.2322	0.0	11327676	3600.5619	1.0

Continued on next page

Table F.1.: Full table of results for the benchmark test

		MILCP-PBB			MILP Reformulation		
Inst.	n	Nodecount	Time	Gap	Nodecount	Time	Gap
128	40	63	1.1429	0.0	4061416	467.6248	0.0
129	40	63	1.2208	0.0	999168	43.8545	0.0
130	40	5755	106.5191	0.0	7951546	3601.1893	0.8
131	40	499	8.2214	0.0	16605223	3600.4410	1.0
132	40	10255	193.0888	0.0	324510	15.9380	0.0
133	40	1895	35.5647	0.0	14613127	3600.3177	inf
134	40	2991	57.7778	0.0	7460113	3601.0309	1.0
135	40	13525	266.4687	0.0	7871520	3600.9640	0.3
136	40	1515	28.0182	0.0	1237433	55.5704	0.0
137	40	9007	169.1595	0.0	8966950	3602.0799	0.5
138	40	7941	146.4297	0.0	9285	0.6387	0.0
139	40	799	13.9483	0.0	10359767	3600.6180	1.0
140	40	611	10.1220	0.0	10538431	3601.7281	1.0
141	40	257	4.3288	0.0	378695	17.4114	0.0
142	40	265	4.2006	0.0	29254	1.6259	0.0
143	40	159	2.4411	0.0	9068492	3601.2563	1.0
144	40	525	9.3265	0.0	17196306	3600.3454	inf
145	40	555	9.4832	0.0	305654	14.3585	0.0
146	40	395	6.7528	0.0	29227	1.5412	0.0
147	40	7181	131.9554	0.0	4123	0.4955	0.0
148	40	111	1.8444	0.0	47	0.0904	0.0
149	40	4481	81.6592	0.0	8360622	3600.0348	0.7
150	40	9257	168.0307	0.0	7472074	3602.0835	0.9
151	40	25387	501.5054	0.0	39936	1.7102	0.0
152	40	1733	31.3487	0.0	7601094	3601.9617	1.0
153	40	683	11.6787	0.0	111478	4.9084	0.0
154	40	2241	40.6295	0.0	10670825	3600.9092	1.0
155	40	973	17.8150	0.0	11087607	3601.9587	1.0
156	40	2875	55.8944	0.0	111054	4.3750	0.0
157	40	157	2.8850	0.0	507267	23.9759	0.0
158	40	997	18.2655	0.0	7339229	3600.6836	0.9
159	40	323	5.5933	0.0	7518166	3601.2365	1.0
160	50	75	2.4067	0.0	118074	7.4862	0.0
161	50	75	2.6881	0.0	15737523	3600.5509	1.0
162	50	93	2.9237	0.0	24167523	1440.7196	0.0
163	50	75	2.4104	0.0	16590792	3600.1574	1.0
164	50	75	2.4451	0.0	85023	5.3421	0.0

Continued on next page

Table F.1.: Full table of results for the benchmark test

		MILCP-PBB			MILP Reformulation		
Inst.	n	Nodecount	Time	Gap	Nodecount	Time	Gap
165	50	77	2.5095	0.0	11243485	3600.6619	1.0
166	50	65	2.2013	0.0	5169	0.6221	0.0
167	50	87	2.7419	0.0	407477	23.7957	0.0
168	50	75	2.6854	0.0	115450	6.4438	0.0
169	50	73	2.5335	0.0	9591538	3600.0894	1.0
170	50	2423	75.2438	0.0	23262723	1353.3727	0.0
171	50	3479	107.8683	0.0	10207243	3600.8648	1.0
172	50	537	16.6234	0.0	8874123	529.2525	0.0
173	50	663	19.9934	0.0	7473436	477.7615	0.0
174	50	19133	602.6309	0.0	3698842	220.2774	0.0
175	50	151	4.6737	0.0	6665548	425.8878	0.0
176	50	10073	324.3787	0.0	13777844	3601.1800	1.0
177	50	813	23.7100	0.0	25622742	3600.7314	1.0
178	50	1941	61.8037	0.0	12217351	767.3749	0.0
179	50	7917	259.3454	0.0	7812356	565.1433	0.0
180	50	517	13.9730	0.0	101603	6.5561	0.0
181	50	669	19.1249	0.0	17008333	3600.3634	1.0
182	50	2479	64.3593	0.0	8264785	3600.8681	1.0
183	50	373	10.5980	0.0	13713250	3601.2056	inf
184	50	761	25.3844	0.0	8354526	3600.5981	1.0
185	50	835	25.7378	0.0	12907715	3600.4083	1.0
186	50	3213	98.3091	0.0	11127819	3600.4181	1.0
187	50	2403	79.7049	0.0	11560675	3600.6479	1.0
188	50	297	8.1076	0.0	10892781	3600.6010	inf
189	50	257	6.8017	0.0	1008120	75.2127	0.0
190	50	7685	251.3642	0.0	26423717	3600.0538	1.0
191	50	11101	371.2965	0.0	10099296	3600.9595	1.0
192	50	211	6.5962	0.0	553920	32.7262	0.0
193	50	21983	736.7869	0.0	15149116	911.1518	0.0
194	50	829	24.8993	0.0	28089011	3600.2808	1.0
195	50	8341	284.3636	0.0	9478323	3600.9178	0.2
196	50	4347	138.9693	0.0	11969616	3600.5600	1.0
197	50	2017	60.5671	0.0	12837013	3600.4998	1.0
198	50	5749	179.4567	0.0	4593546	321.5780	0.0
199	50	3761	118.9267	0.0	24535069	3600.0516	1.0
200	60	95	5.3018	0.0	24886131	3600.0699	inf
201	60	97	4.8789	0.0	39455825	3600.3901	1.0

Continued on next page

Table F.1.: Full table of results for the benchmark test

		MILCP-PBB			MILP Reformulation		
Inst.	n	Nodecount	Time	Gap	Nodecount	Time	Gap
202	60	89	5.0732	0.0	50259520	3600.0707	1.0
203	60	91	4.9021	0.0	24262141	3600.0810	inf
204	60	87	4.6540	0.0	6616978	476.7363	0.0
205	60	109	5.3670	0.0	35401147	2576.5431	0.0
206	60	119	5.9437	0.0	40436992	3600.3677	1.0
207	60	93	4.7145	0.0	19520919	3600.2277	1.0
208	60	89	4.5565	0.0	16021670	3600.3017	1.0
209	60	93	4.7323	0.0	19018050	3600.1198	inf
210	60	6037	295.2125	0.0	50200819	3600.0655	1.0
211	60	2041	95.7751	0.0	46197987	3600.0643	1.0
212	60	2967	146.5233	0.0	55507819	3600.0645	1.0
213	60	14517	742.8341	0.0	8752981	3600.0631	0.8
214	60	34181	1887.6959	0.0	51471242	3600.0702	1.0
215	60	61740	3600.0797	1.0	28888396	3600.1671	1.0
216	60	12109	613.7741	0.0	49749191	3600.0639	1.0
217	60	61843	3600.0101	1.0	41910105	3600.0652	0.8
218	60	60029	3600.0041	1.0	34435549	3600.0681	1.0
219	60	6487	313.5253	0.0	50485915	3600.0652	1.0
220	60	2065	92.3607	0.0	37154426	3600.0675	1.0
221	60	1329	58.2435	0.0	51978365	3600.0618	1.0
222	60	2199	102.9947	0.0	49909484	3600.0607	1.0
223	60	3605	173.2250	0.0	13798637	3600.8346	inf
224	60	35191	1930.7522	0.0	46423252	3600.0649	1.0
225	60	449	19.2751	0.0	12828170	3600.0870	1.0
226	60	4581	241.6146	0.0	14370212	3600.2924	1.0
227	60	1707	72.3106	0.0	12025957	3600.5277	1.0
228	60	857	42.0910	0.0	10572822	3600.0628	1.0
229	60	447	19.0982	0.0	47778010	3600.0644	1.0
230	60	9109	480.4296	0.0	46400262	3600.0838	1.0
231	60	12591	646.9530	0.0	44378091	3600.1040	1.0
232	60	28897	1657.8787	0.0	37531820	3600.6144	1.0
233	60	17657	965.3765	0.0	45138783	3600.0623	1.0
234	60	64886	3600.0059	1.0	48157788	3600.0694	1.0
235	60	35649	1872.9085	0.0	46689688	3600.0763	1.0
236	60	64875	3600.0146	1.0	31122179	3600.1779	1.0
237	60	21371	1109.2915	0.0	34486366	3600.4863	1.0
238	60	405	20.2924	0.0	46589263	3600.1261	1.0

Continued on next page

Table F.1.: Full table of results for the benchmark test

		MILCP-PBB			MILP Reformulation		
Inst.	n	Nodecount	Time	Gap	Nodecount	Time	Gap
239	60	68694	3600.0189	1.0	46161473	3438.6199	0.0
240	70	109	8.6188	0.0	42567729	3600.0726	1.0
241	70	113	9.8997	0.0	38874776	3600.0845	1.0
242	70	105	8.8139	0.0	39860152	3600.1576	1.0
243	70	123	9.8650	0.0	38456038	3600.5312	1.0
244	70	111	8.9915	0.0	42911066	3600.1150	1.0
245	70	93	7.2621	0.0	40853430	3600.1202	1.0
246	70	129	10.2514	0.0	36640641	3600.0826	1.0
247	70	93	7.6490	0.0	24608285	3600.3091	1.0
248	70	101	8.0807	0.0	41219545	3600.0820	1.0
249	70	111	10.0252	0.0	45596393	3600.0840	1.0
250	70	47358	3600.0651	1.0	40576274	3600.1004	1.0
251	70	42187	3255.0700	0.0	18265067	3600.0882	1.0
252	70	9081	643.1961	0.0	41919912	3600.0926	1.0
253	70	5279	371.7403	0.0	39359918	3600.1038	1.0
254	70	45762	3600.0458	1.0	40670051	3600.1080	1.0
255	70	5639	395.7295	0.0	43996932	3600.1232	1.0
256	70	48895	3600.0370	1.0	40814135	3600.0926	1.0
257	70	47809	3600.0449	1.0	39977813	3600.1089	1.0
258	70	35707	2721.0497	0.0	40754231	3600.0940	1.0
259	70	14121	1025.1795	0.0	43597128	3600.0873	1.0
260	70	4375	334.7261	0.0	42817797	3600.0870	1.0
261	70	41545	3427.4645	0.0	31341882	3600.1065	1.0
262	70	34339	2705.8331	0.0	35895136	3600.1102	1.0
263	70	10931	846.5420	0.0	32521586	3600.0902	1.0
264	70	15677	1194.4300	0.0	42718678	3600.1043	1.0
265	70	3709	275.7832	0.0	21751573	3600.3133	1.0
266	70	5535	383.8608	0.0	40206796	3600.0844	1.0
267	70	7121	566.9860	0.0	42861673	3600.1063	1.0
268	70	2389	166.9705	0.0	41814757	3600.0814	1.0
269	70	4003	259.0037	0.0	27686228	3600.0991	1.0
270	70	46394	3600.0546	1.0	41778529	3600.0901	1.0
271	70	47689	3600.0304	1.0	38073755	3600.0906	1.0
272	70	4223	297.0891	0.0	41609098	3600.1751	1.0
273	70	30317	2217.2765	0.0	39343286	3600.0843	1.0
274	70	46016	3600.0892	1.0	41741473	3600.0845	1.0
275	70	46335	3600.0094	1.0	41790802	3600.0937	1.0

Continued on next page

Table F.1.: Full table of results for the benchmark test

		MILCP-PBB			MILP Reformulation		
Inst.	n	Nodecount	Time	Gap	Nodecount	Time	Gap
276	70	27863	2097.7150	0.0	40397771	3600.0882	1.0
277	70	6817	492.6976	0.0	41169034	3600.1027	1.0
278	70	48204	3600.0415	1.0	34593215	3600.1144	1.0
279	70	17457	1258.1579	0.0	44017761	3600.1165	1.0
280	80	105	11.2785	0.0	37048535	3600.1103	inf
281	80	127	14.0814	0.0	33118169	3600.1055	1.0
282	80	133	14.6090	0.0	33375113	3600.1555	1.0
283	80	123	13.2798	0.0	34446499	3600.1007	1.0
284	80	123	13.1252	0.0	21163471	3600.1148	1.0
285	80	151	16.0503	0.0	36012728	3600.1113	1.0
286	80	97	10.8823	0.0	33777922	3600.1117	1.0
287	80	119	13.9151	0.0	33245391	3600.4252	1.0
288	80	135	15.6705	0.0	36896396	3600.1092	1.0
289	80	93	10.0849	0.0	36262101	3600.1025	1.0
290	80	1607	153.6416	0.0	36320027	3600.1353	1.0
291	80	33824	3600.0716	1.0	37148555	3600.1075	1.0
292	80	19783	1968.2480	0.0	37963021	3600.1055	1.0
293	80	35375	3600.0942	1.0	36263490	3600.1171	inf
294	80	19409	1938.5354	0.0	36041135	3600.1160	1.0
295	80	34775	3600.0294	1.0	35651234	3600.1093	1.0
296	80	9091	891.2372	0.0	22816802	3600.1098	1.0
297	80	33987	3600.0496	1.0	32713852	3600.1184	1.0
298	80	34070	3600.0215	1.0	37148706	3600.1194	1.0
299	80	35236	3600.0739	1.0	36150335	3600.1072	1.0
300	80	34368	3600.0083	1.0	37044501	3600.1438	1.0
301	80	34145	3600.1220	1.0	12718257	3600.1119	1.0
302	80	11463	1262.4349	0.0	35606393	3600.1354	1.0
303	80	29973	3376.4791	0.0	35082738	3600.1228	1.0
304	80	35710	3600.0329	1.0	37562327	3600.1011	1.0
305	80	6069	575.2140	0.0	33561958	3600.1135	1.0
306	80	16279	1767.7997	0.0	30763584	3600.1085	inf
307	80	32377	3600.1488	1.0	36400912	3600.1512	1.0
308	80	34040	3600.0696	1.0	33662507	3600.1395	1.0
309	80	32192	3600.0694	1.0	36859114	3600.1240	1.0
310	80	34455	3530.8680	0.0	34617513	3600.1253	1.0
311	80	34579	3600.1159	1.0	29326655	3600.1057	1.0
312	80	32611	3600.1423	1.0	37007489	3600.1540	inf

Continued on next page

Table F.1.: Full table of results for the benchmark test

		MILCP-PBB			MILP Reformulation		
Inst.	n	Nodecount	Time	Gap	Nodecount	Time	Gap
313	80	33103	3600.1624	1.0	35514103	3600.1269	1.0
314	80	28505	2924.3102	0.0	33926880	3600.1404	inf
315	80	35539	3600.2558	1.0	35104508	3600.1748	1.0
316	80	32425	3290.5276	0.0	28700960	3600.1290	inf
317	80	33489	3600.0685	1.0	37450092	3600.1244	inf
318	80	34421	3600.1052	1.0	35102244	3600.1242	1.0
319	80	33877	3600.3986	1.0	33193558	3600.1392	inf
320	90	127	19.5232	0.0	16937471	3603.4166	1.0
321	90	153	22.3891	0.0	29376766	3600.1407	1.0
322	90	155	22.8672	0.0	28400929	3600.1446	1.0
323	90	103	16.2990	0.0	11030656	1310.1262	0.0
324	90	129	20.3978	0.0	15280804	3600.1487	1.0
325	90	117	17.7087	0.0	31528912	3600.1777	1.0
326	90	157	22.7151	0.0	32774609	3600.1671	inf
327	90	147	22.2076	0.0	9583300	3600.1760	inf
328	90	135	20.6060	0.0	15050902	3600.1643	inf
329	90	147	21.0489	0.0	7349433	3600.1545	inf
330	90	24160	3600.7680	1.0	30578988	3600.1763	inf
331	90	24248	3600.0638	1.0	30752831	3600.1825	inf
332	90	23512	3600.4164	1.0	10330518	3600.1849	inf
333	90	24846	3600.3358	1.0	14783669	3600.1461	inf
334	90	22740	3600.5143	1.0	11384023	3600.1609	inf
335	90	24184	3600.4909	1.0	22889268	3600.1467	inf
336	90	23297	3600.1440	1.0	27435194	3600.1436	inf
337	90	23550	3600.7915	1.0	25139388	3600.1628	1.0
338	90	23068	3600.4417	1.0	26181117	3600.1376	1.0
339	90	22413	3600.2280	1.0	13667703	3600.1619	inf
340	90	4113	438.4722	0.0	12949837	3600.1507	inf
341	90	7999	1049.6061	0.0	11562773	3600.1869	1.0
342	90	19529	3600.0214	1.0	24331461	3600.1389	1.0
343	90	20653	3600.0472	1.0	24869728	3600.1354	inf
344	90	1129	114.3678	0.0	23005084	3600.1628	1.0
345	90	18032	3601.1323	1.0	13836078	3600.1692	inf
346	90	5741	678.3576	0.0	21154208	3600.1392	1.0
347	90	18233	3600.6711	1.0	21596076	3600.1652	inf
348	90	16234	3600.5550	1.0	19934197	3600.1908	inf
349	90	15311	3600.1962	1.0	11888119	3600.1430	inf

Continued on next page

Table F.1.: Full table of results for the benchmark test

		MILCP-PBB			MILP Reformulation		
Inst.	n	Nodecount	Time	Gap	Nodecount	Time	Gap
350	90	11055	1464.3901	0.0	12307174	3600.1554	1.0
351	90	11402	3600.1947	1.0	12361298	3600.1701	inf
352	90	10968	3600.3004	1.0	12514407	3600.1754	inf
353	90	9537	3600.0392	1.0	10986432	3600.1394	inf
354	90	8693	3600.1438	1.0	9489251	3600.1425	inf
355	90	7906	3600.2449	1.0	3203258	3600.1678	inf
356	90	6263	3600.2285	1.0	4052997	3600.1839	inf
357	90	4819	3600.8998	1.0	4090435	3600.2016	inf
358	90	4051	3600.7764	1.0	3850978	3600.1976	inf
359	90	3741	3600.7337	1.0	3704230	3600.3116	inf
360	100	133	206.8423	0.0	3770082	3600.3518	inf
361	100	149	213.4820	0.0	1837594	3600.3681	inf
362	100	119	192.6571	0.0	3692026	3600.3320	1.0
363	100	175	230.3095	0.0	3920317	3600.3986	inf
364	100	173	249.1848	0.0	3697418	3600.4362	inf
365	100	159	225.3572	0.0	3612315	3600.4405	inf
366	100	157	229.9255	0.0	3450451	3600.4868	inf
367	100	141	206.5451	0.0	409605	298.1112	0.0
368	100	159	233.7223	0.0	3653556	3600.5653	inf
369	100	165	238.6822	0.0	2221500	3600.5923	inf
370	100	2328	3601.1221	1.0	2832712	3600.2091	1.0
371	100	2418	3600.0912	1.0	1867499	3600.4453	inf
372	100	2378	3600.7563	1.0	3899805	3600.6171	1.0
373	100	2150	3600.7655	1.0	1403491	3600.8033	inf
374	100	2162	3600.2248	1.0	1544125	3600.4468	inf
375	100	2141	3601.6171	1.0	3871515	3601.0273	inf
376	100	2176	3600.0478	1.0	3155588	3600.9158	inf
377	100	2459	3600.8661	1.0	2568930	3601.0041	inf
378	100	2345	3600.2753	1.0	3077182	3600.7871	inf
379	100	2459	3600.4004	1.0	2729111	3600.8324	inf
380	100	2753	3600.6766	1.0	2312640	3600.8802	inf
381	100	2363	3600.3465	1.0	2398795	3600.5612	inf
382	100	3192	3600.7711	1.0	3093638	3600.6241	inf
383	100	3158	3600.8031	1.0	2008216	3600.6386	inf
384	100	2400	3600.3298	1.0	3613305	3600.6398	inf
385	100	2576	3600.5092	1.0	1958350	3600.3754	inf
386	100	3343	3600.6137	1.0	4008430	3600.2511	inf

Continued on next page

Table F.1.: Full table of results for the benchmark test

		MILCP-PBB			MILP Reformulation		
Inst.	n	Nodecount	Time	Gap	Nodecount	Time	Gap
387	100	2277	3600.6431	1.0	3442692	3600.9500	1.0
388	100	3267	3600.2265	1.0	1882170	3600.7430	inf
389	100	2487	3600.5274	1.0	3965266	3600.9704	inf
390	100	2420	3600.5672	1.0	3484015	3601.0077	inf
391	100	2417	3600.1276	1.0	2975610	3600.9602	inf
392	100	2315	3600.1726	1.0	2096589	3600.6541	inf
393	100	2690	3600.5409	1.0	1736041	3601.3589	inf
394	100	2790	3600.3883	1.0	2854415	3600.7975	inf
395	100	2867	3600.1787	1.0	2518644	3601.0965	inf
396	100	3495	3600.0890	1.0	5510967	3601.0042	inf
397	100	4203	3600.1501	1.0	7398050	3601.3565	inf
398	100	8434	3600.0030	1.0	9420922	3601.1482	inf
399	100	8911	3600.0430	1.0	10020329	3600.6226	inf

Appendix G

Tables of Aggregated Results of Other Settings for MILP

In what follows, we include all tables for the aggregated running times and node counts of the settings not reported in Section 5.3.

G.1. Branching Rule Test

Table G.1.: Aggregated nodecounts (top) and runtimes (bottom) for the branching rule test with random choice

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
≤ 200	18625.0	**1127.0**	37.0	156225.0	**41.2**	**49.6**	2266.7	14
≤ 500	10206.3	6255.0	137.0	35297.0	138.2	140.8	2962.3	14
≤ 2000	52138.8	4514.5	**5.0**	199293.0	**5.1**	**5.3**	3268.1	15
≤ 5000	10252.3	1945.0	55.0	28757.0	64.4	83.4	1991.3	**43**
> 5000	66.5	33.0	17.0	183.0	17.0	17.1	55.3	11
≤ 200	324.8	9.6	**0.0**	2625.5	**0.0**	0.0	35.1	14
≤ 500	94.4	85.6	1.0	227.1	1.0	1.0	44.8	14
≤ 2000	2982.5	45.5	0.1	10242.5	**0.1**	**0.1**	146.0	15
≤ 5000	350.7	92.6	4.7	954.8	5.1	6.0	103.4	**43**
> 5000	519.6	11.0	1.5	2055.2	1.5	1.7	46.8	11

Table G.2.: Aggregated nodecounts (top) and runtimes (bottom) for the branching rule test with pseudocost branching

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
≤ 200	**11614.5**	2187.0	45.0	113727.0	49.0	57.0	2103.1	**19**
≤ 500	30660.0	13971.0	1303.0	94761.0	1305.3	1309.8	10885.4	**16**
≤ 2000	20000.8	9420.5	7.0	77791.0	7.2	7.6	2607.1	10
≤ 5000	2573.7	3357.0	33.0	4331.0	49.6	82.9	1167.7	**43**
> 5000	123.0	91.0	**3.0**	307.0	**3.0**	3.1	87.4	11
≤ 200	**260.7**	13.2	**0.0**	**2400.6**	**0.0**	**0.0**	29.7	**19**
≤ 500	608.5	238.6	11.1	2194.9	11.2	11.2	181.2	**16**
≤ 2000	1054.6	119.6	0.2	4562.4	0.2	0.2	125.7	10
≤ 5000	201.9	171.9	3.1	430.7	4.0	5.7	91.7	**43**
> 5000	702.7	81.3	1.5	2646.6	1.5	1.5	80.1	11

Table G.3.: Aggregated nodecounts (top) and runtimes (bottom) for the branching rule test with MILP-based branching

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
≤ 200	14379.1	2010.0	**43.0**	**109661.0**	50.1	64.1	2198.6	14
≤ 500	10123.8	6103.0	**119.0**	33204.0	**119.1**	**119.2**	**1986.7**	**16**
≤ 2000	**16784.8**	**2323.5**	**5.0**	**53861.0**	6.6	10.0	**1942.6**	**19**
≤ 5000	1126.7	1633.0	74.0	1673.0	81.8	97.4	711.6	40
> 5000	28.5	**7.0**	**3.0**	97.0	**3.0**	**3.0**	**23.4**	**13**
≤ 200	268.6	**5.9**	**0.0**	2605.0	**0.0**	**0.0**	27.5	14
≤ 500	86.3	67.3	**0.9**	204.3	**0.9**	**0.9**	36.0	**16**
≤ 2000	**476.9**	**15.1**	0.1	**1440.8**	**0.1**	0.2	**59.2**	**19**
≤ 5000	40.1	36.3	1.9	81.9	**2.1**	**2.5**	27.0	40
> 5000	241.6	**1.8**	0.5	962.2	**0.5**	**0.5**	**24.5**	**13**

G.2. Node Selection Test

Table G.4.: Aggregated nodecounts (top) and runtimes (bottom) for the node selection test with breadth-first search

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
≤ 200	9125.2	1375.5	21.0	87809.0	22.1	24.4	1612.6	**16**
≤ 500	24475.0	8937.0	169.0	114935.0	170.6	173.8	**3962.5**	**14**
≤ 2000	23867.1	10168.0	7.0	**108761.0**	7.3	7.8	3136.4	15
≤ 5000	746.0	**746.0**	29.0	**1463.0**	32.6	39.8	349.0	**42**
> 5000	225.6	**27.0**	**3.0**	1026.0	**3.1**	**3.2**	**92.3**	11
≤ 200	215.3	5.7	**0.0**	3544.9	**0.0**	**0.0**	18.0	**16**
≤ 500	988.0	117.0	**1.1**	5616.4	**1.1**	1.2	102.3	**14**
≤ 2000	1864.6	205.7	0.2	9564.2	0.2	0.2	181.3	15
≤ 5000	32.8	32.8	2.1	**63.5**	2.2	2.6	19.8	**42**
> 5000	273.6	**11.3**	**1.0**	873.9	**1.0**	**1.0**	54.7	11

Table G.5.: Aggregated nodecounts (top) and runtimes (bottom) for the node selection test with depth-first search

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
≤ 200	**6668.7**	**1192.0**	**11.0**	**72279.0**	**12.6**	**15.8**	**1330.0**	**16**
≤ 500	**10975.9**	**5871.0**	663.0	**34971.0**	672.0	690.1	5454.0	**14**
≤ 2000	**19519.7**	**1936.0**	**5.0**	117391.0	**5.0**	**5.1**	**1642.4**	12
≤ 5000	**773.0**	773.0	**21.0**	1525.0	**24.8**	**32.3**	**343.4**	40
> 5000	**174.8**	145.0	25.0	**428.0**	25.0	25.0	134.5	14
≤ 200	**62.2**	**4.0**	**0.0**	**862.0**	**0.0**	**0.0**	**11.7**	**16**
≤ 500	**152.9**	**51.8**	4.1	**529.5**	4.1	4.3	**64.9**	**14**
≤ 2000	1057.7	**30.0**	**0.1**	6287.4	**0.1**	**0.1**	**78.1**	12
≤ 5000	**32.6**	**32.6**	**1.5**	63.7	**1.7**	**2.0**	**19.2**	40
> 5000	**132.2**	20.4	9.1	**561.1**	9.1	9.2	**45.0**	14

G.3. Warmstart Test

Table G.6.: Aggregated nodecounts (top) and runtimes (bottom) for the warmstart test using PStart/DStart

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
≤ 200	16147.0	1635.0	**15.0**	173943.0	**16.4**	**19.3**	1934.4	14
≤ 500	16751.3	8185.0	373.0	69717.0	374.4	377.3	5439.8	14
≤ 2000	32875.0	3885.0	**7.0**	123723.0	24.5	59.4	3556.0	15
≤ 5000	1434.0	1434.0	41.0	2827.0	48.0	61.9	542.4	8
> 5000	1487.4	441.0	93.0	7551.0	96.8	104.3	654.2	**40**
≤ 200	219.8	7.9	**0.0**	3165.8	**0.0**	**0.0**	17.8	14
≤ 500	429.1	204.7	4.0	1518.9	4.1	4.1	144.0	14
≤ 2000	1933.1	177.8	0.2	7376.5	1.1	2.8	210.8	15
≤ 5000	102.0	102.0	7.5	196.4	8.0	8.9	50.1	8
> 5000	2499.6	1704.1	77.3	8449.3	77.5	77.7	769.2	**40**

G.4. Valid Inequalities Test

Table G.7.: Aggregated nodecounts (top) and runtimes (bottom) for the valid inequalities test with all simple cuts

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
≤ 200	15391.7	1621.0	19.0	159447.0	**20.4**	**23.2**	2407.4	**40**
≤ 500	**20164.5**	19676.0	163.0	**41143.0**	200.5	275.5	**6552.1**	**10**
≤ 2000	**43997.4**	**9567.0**	7.0	169423.0	**44.7**	**120.2**	8203.7	8
≤ 5000	**1378.0**	**1378.0**	41.0	**2715.0**	**47.7**	**61.1**	**530.0**	11
> 5000	1204.9	405.0	51.0	**7583.0**	**51.7**	**53.1**	411.2	**17**
≤ 200	352.0	8.2	**0.0**	**5160.9**	**0.0**	**0.0**	25.2	**40**
≤ 500	**367.8**	**316.2**	1.8	**836.9**	2.1	2.9	**127.9**	**10**
≤ 2000	**2414.4**	159.2	**0.1**	**7835.4**	**0.8**	**2.2**	350.2	8
≤ 5000	59.0	59.0	**3.0**	114.9	**3.3**	**3.9**	30.3	11
> 5000	1206.3	**326.6**	**24.7**	3274.7	**25.1**	**25.9**	349.4	**17**

G.5. Upper Bound Test

Table G.8.: Aggregated nodecounts (top) and runtimes (bottom) for the upper bound test for MILP-PBB and warm-starts off

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
≤ 200	57.9	**1.0**	**1.0**	969.0	**1.0**	**1.0**	16.0	36
≤ 500	2.6	**1.0**	**1.0**	43.0	**1.0**	**1.0**	2.4	44
≤ 2000	1.2	**1.0**	**1.0**	5.0	**1.0**	**1.0**	1.2	58
≤ 5000	8910.5	**1.0**	**1.0**	53458.0	**1.0**	**1.0**	187.3	33
> 5000	**1.0**	**1.0**	**1.0**	1.0	**1.0**	**1.0**	**1.0**	42
≤ 200	**0.0**	**0.0**	**0.0**	**0.3**	**0.0**	**0.0**	**0.0**	36
≤ 500	**0.0**	**0.0**	**0.0**	**0.2**	**0.0**	**0.0**	**0.0**	44
≤ 2000	**0.0**	**0.0**	**0.0**	**0.5**	**0.0**	**0.0**	**0.0**	58
≤ 5000	**0.1**	**0.1**	**0.0**	**0.1**	**0.0**	**0.0**	**0.1**	33
> 5000	**1.8**	**2.5**	**0.1**	**3.8**	**0.1**	**0.1**	**1.7**	42

Table G.9.: Aggregated nodecounts (top) and runtimes (bottom) for the upper bound test for MILP-PBB with warmstarts using VBasis/CBasis

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
≤ 200	89.4	**1.0**	**1.0**	1503.0	**1.0**	**1.0**	18.8	38
≤ 500	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	44
≤ 2000	10.4	**1.0**	**1.0**	255.0	**1.0**	**1.0**	5.8	60
≤ 5000	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	29
> 5000	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	48
≤ 200	0.1	**0.0**	**0.0**	1.3	**0.0**	**0.0**	**0.1**	38
≤ 500	0.0	**0.0**	**0.0**	0.2	**0.0**	**0.0**	**0.0**	44
≤ 2000	0.3	**0.0**	**0.0**	7.9	**0.0**	**0.0**	0.3	60
≤ 5000	**0.1**	**0.1**	**0.0**	0.3	**0.0**	**0.0**	**0.1**	29
> 5000	2.0	2.9	**0.1**	4.0	**0.1**	**0.1**	1.9	48

G.6. Benchmark Test

Table G.10.: Aggregated nodecounts (top) and runtimes (bottom) for the benchmark test for MILP-PBB without warmstarts

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
≤ 200	15121.8	1506.0	19.0	153407.0	20.3	23.0	2032.8	14
≤ 500	17443.0	7997.0	147.0	69707.0	148.5	151.4	4911.1	10
≤ 2000	40426.2	3682.5	7.0	121235.0	7.1	7.2	3506.6	17
≤ 5000	1378.0	1378.0	41.0	2715.0	47.7	61.1	530.0	40
> 5000	1194.4	301.0	51.0	7583.0	51.7	53.2	387.1	16
≤ 200	288.0	4.7	0.1	4408.1	0.1	0.1	17.0	14
≤ 500	301.7	80.7	0.9	1338.3	0.9	0.9	85.8	10
≤ 2000	1785.5	69.6	**0.1**	7259.6	**0.1**	**0.1**	146.9	17
≤ 5000	60.1	60.1	3.1	117.1	3.3	3.9	30.7	40
> 5000	951.2	268.6	20.7	2503.1	21.0	21.6	270.1	16

Table G.11.: Aggregated nodecounts (top) and runtimes (bottom) for the benchmark test for MILP-PBB with warm-starts using VBasis/CBasis

n	Avg.	Med.	Min.	Max.	$Q(0.25)$	$Q(0.75)$	GSM	Perc.
≤ 200	14900.4	1285.0	19.0	149435.0	20.2	22.7	1673.8	16
≤ 500	24306.1	4053.0	209.0	113939.0	209.9	211.8	3616.1	19
≤ 2000	30071.7	2181.0	7.0	122289.0	7.1	7.2	1988.5	17
≤ 5000	735.0	735.0	31.0	1439.0	34.5	41.6	349.0	38
> 5000	292.6	178.5	5.0	**1129.0**	5.1	5.3	169.9	10
≤ 200	643.1	3.9	**0.0**	7566.8	**0.0**	**0.0**	19.4	16
≤ 500	1030.2	47.8	1.7	5688.0	1.7	1.7	97.4	19
≤ 2000	1956.1	73.6	0.2	9550.8	0.2	0.2	115.9	17
≤ 5000	37.4	37.4	3.9	70.8	4.1	4.4	23.5	38
> 5000	934.6	172.1	1.6	6356.8	1.6	1.6	120.1	10

Appendix H

Full Results for the Benchmark Test for MILP

Here, we present running times, node counts, and optimality gaps for the benchmark test in Section 5.3.2. As both versions of MILP-PBB are rather similar, we only include the first versions, as it solved slightly more instances.

Table H.1.: Full table of results for the benchmark test

	MILP-PBB 1			Gurobi		
Inst.	Nodecount	Time	Gap	Nodecount	Time	Gap
22433	10171	151.8781	0.0	23	1.0669	0.0
23588	7997	80.6618	0.0	779	1.7898	0.0
2club200v15p5scn	5228	10800.0169	1.0	37076	10800.1420	0.9
a1c1s1	137636	10800.0181	0.9	1891453	10800.1686	0.7
a2c1s1	141081	10800.0656	1.0	1809562	10800.1550	0.7
aflow30a	205652	10800.0261	0.2	40925	62.2703	0.0
air01	7	0.1431	0.0	3	0.1795	0.0
air02	467	40.7614	0.0	5	2.3438	0.0
air03	161	37.1691	0.0	3	6.0587	0.0
air04	21704	10800.0009	0.0	546	85.8062	0.0
air05	61397	10800.0007	0.0	701	44.7843	0.0
air06	51	20.6997	0.0	8	8.4369	0.0
app3	2715	117.0755	0.0	745	2.9346	0.0
assign1-5-8	230783	10800.0461	0.1	7394564	10800.0407	0.1
b-ball	234080	10800.0401	0.2	73916387	10800.0411	0.2
b1c1s1	134944	10800.1451	1.0	617748	10800.1890	0.5

Continued on next page

Table H.1.: Full table of results for the benchmark test

	MILP-PBB 1			Gurobi		
Inst.	Nodecount	Time	Gap	Nodecount	Time	Gap
b2c1s1	127503	10800.1239	0.9	321486	10800.1831	0.5
bc	63563	10800.0464	0.9	18886	10800.3304	0.8
bc1	63610	10800.0007	0.7	78499	10800.2930	0.2
beavma	215335	10800.0136	0.8	3282	1.8216	0.0
bell3a	239	29.0565	0.0	8691	2.2281	0.0
bell3b	1123	10800.0003	0.0	8037	2.1618	0.0
bell4	216	7201.2302	0.2	23286	5.1704	0.0
bell5	1029	533.9247	0.0	9889	1.7574	0.0
bg512142	185356	10800.0492	1.0	1193795	10800.0702	0.2
bienst1	113937	7259.5778	0.0	10554	54.0371	0.0
bienst2	146903	10800.0010	0.8	77430	275.4696	0.0
binkar10_1	180654	10800.0640	0.3	2307883	5323.7408	0.0
bm23	1497	1.3843	0.0	255	0.0360	0.0
control20-5-10-5	1	0.0516	inf	1	0.1550	inf
control30-3-2-3	216297	10800.0076	inf	20752642	10800.0393	inf
control30-5-10-4	1	0.0696	inf	1	0.2158	inf
cost266-UUE	131011	10800.0516	0.7	1209038	10800.3638	0.0
cov1075	153133	10800.0751	0.1	624928	10801.1858	0.0
cracpb1	13	0.1842	0.0	3	0.1501	0.0
dano3_3	93	2175.5454	0.0	29	67.1687	0.0
dano3_5	583	10800.0036	0.0	341	790.8147	0.0
danoint	103983	10800.0383	0.1	966432	10800.0380	0.0
dcmulti	169097	6571.9724	0.0	1020	0.9485	0.0
dell	59921	6891.2883	5.2	4	0.1464	inf
diamond	0	0.0014	inf	1	0.0030	inf
dsbmip	7631	1961.3982	0.0	10	0.3505	0.0
egout	297810	10800.0254	0.7	1506	0.2326	0.0
enigma	5651	14.7379	0.0	146	0.0594	0.0
eva1aprime5x5opt	2400	10800.0038	164.4	3181	10800.1365	193.3
exp-1-500-5-5	212066	10800.0843	0.7	19763983	10800.0542	0.3
f2gap40400	69707	1338.3264	0.0	400	0.1813	0.0
fastxgemm-n2r6s0t2	56015	10800.0053	1.0	167007	9505.3164	0.0
fastxgemm-n2r7s4t1	55199	10800.0009	1.0	131785	10800.1075	0.4
fastxgemm-n3r21s3t6	4	10800.4050	1.0	1	10801.6766	inf
fastxgemm-n3r22s4t6	4	10800.0449	1.0	1	10801.6964	inf
fastxgemm-n3r23s5t6	4	10800.0821	1.0	1	10801.8025	inf
fhnw-binpack4-18	1	10800.0530	1.0	6893887	10800.0300	inf

Continued on next page

Table H.1.: Full table of results for the benchmark test

Inst.	MILP-PBB 1 Nodecount	Time	Gap	Gurobi Nodecount	Time	Gap
fhnw-binpack4-4	1	10800.0487	1.0	9074868	10800.0357	inf
fixnet3	327476	10800.0155	0.5	24522476	10800.0921	0.1
fixnet4	395948	10800.0319	0.7	26538150	10800.0291	0.1
fixnet6	236396	10800.0260	0.8	15674807	10800.1534	0.1
flugpl	1	0.0065	0.0	251	0.0157	0.0
g503inf	12	0.0664	1.1	4	0.0130	inf
gen	10231	1048.3965	0.0	132	0.3677	0.0
glass-sc	38352	10800.0104	0.4	83349	10800.0857	0.1
glass4	229798	10800.1067	0.6	11479346	10800.0326	0.3
go19	213398	10800.0732	0.1	1013315	10801.6447	0.0
gr4x6	273	0.2732	0.0	32	0.0115	0.0
gsvm2rl11	28806	10800.0397	0.9	17027	10800.3312	0.9
gsvm2rl12	6054	10800.0041	1.0	12660	10800.3276	0.9
gsvm2rl3	197849	10800.0072	1.0	8523737	10800.0382	0.1
gsvm2rl5	185786	10800.0216	1.0	2425813	10800.0461	0.6
gsvm2rl9	83623	10800.0312	1.0	125313	10800.0741	0.8
iis-100-0-cov	75100	10800.0026	0.6	141932	7530.3781	0.0
iis-bupa-cov	58717	10800.0047	0.4	108885	10800.2832	0.1
iis-glass-cov	36066	10800.0172	0.4	32232	4915.9059	0.0
iis-hc-cov	13652	10800.0178	0.4	31485	10800.1438	0.1
istanbul-no-cutoff	7761	10800.0044	0.7	2816	1003.5724	0.0
k16x240	258770	10800.0301	0.7	24557027	10802.2853	0.2
k16x240b	220215	10800.0980	0.7	15123397	10800.0283	0.3
khb05250	9567	149.4044	0.0	1040	0.7205	0.0
l152lav	121235	3313.9982	0.0	297	1.6368	0.0
lp4l	2609	50.9222	0.0	25	0.3024	0.0
lseu	82553	1107.4340	0.0	2358	0.3670	0.0
m100n500k4r1	202884	10800.1078	0.0	924074	2537.0217	0.0
mad	207331	10800.1407	1.0	7272093	8199.7980	0.0
map06	342	10800.0271	61.0	527	10805.6785	0.3
map10	434	10800.0389	2.8	728	10805.0429	0.2
map14	1365	10800.0661	5.9	936	10807.4080	0.1
map14860-20	1072	10800.4300	0.9	1426	10805.1722	0.0
map16715-04	367	10800.1454	295.3	596	10805.7876	0.8
map18	975	10800.0838	0.1	1401	10805.3596	0.0
map20	1288	10800.0393	0.8	1865	10807.4756	0.0
markshare1	214645	10800.0008	1.0	41883775	10800.0190	1.0

Continued on next page

Table H.1.: Full table of results for the benchmark test

	MILP-PBB 1			Gurobi		
Inst.	Nodecount	Time	Gap	Nodecount	Time	Gap
markshare2	212938	10800.0550	1.0	38530808	10800.0209	1.0
markshare_4_0	224973	10800.0242	1.0	228246	22.7696	0.0
markshare_5_0	216190	10800.0654	1.0	39018139	8037.3730	0.0
mas74	207749	10800.1107	0.2	2821861	980.8020	0.0
mas76	218691	10800.0825	0.1	329883	70.3494	0.0
milo-v13-4-3d-3-0	136331	10800.0084	0.9	1042738	10800.0198	inf
milo-v13-4-3d-4-0	105430	10800.0305	0.9	516519	10800.0367	inf
misc01	1453	3.0617	0.0	422	0.1371	0.0
misc02	263	0.3762	0.0	53	0.0351	0.0
misc03	1515	6.3611	0.0	1099	0.7762	0.0
misc04	41	3.0556	0.0	12	0.5964	0.0
misc04inf	11	33.2024	1.0	4	0.5915	inf
misc05	1931	10.2001	0.0	227	0.2454	0.0
misc05inf	225	4.4494	1.0	22	0.0813	inf
misc06	3059	88.2802	0.0	47	0.3287	0.0
misc07	28283	488.6593	0.0	24277	32.7908	0.0
mod008	326647	10800.0492	0.1	4533	0.8086	0.0
mod008inf	3947	227.0458	1.0	414	0.1007	inf
mod010	289836	10800.0301	0.0	137	1.0603	0.0
mod011	30970	10800.1014	0.1	4517	167.8306	0.0
mod013	1105	1.9309	0.0	256	0.0518	0.0
modglob	230928	10800.0086	0.1	1126011	707.6447	0.0
neos-1122047	1602	10800.0294	0.0	1	9.1809	0.0
neos-1396125	104140	10800.0167	1.0	6380	182.6387	0.0
neos-1426635	209328	10800.0570	0.0	6773125	10800.4927	0.0
neos-1426662	191618	10800.0458	0.2	1182851	10800.5453	0.2
neos-1430701	173997	10800.0854	0.0	288699	1151.6380	0.0
neos-1436709	118396	4917.8889	0.0	1466328	10800.9059	0.0
neos-1440460	192242	10800.0143	0.0	2512389	10800.5153	0.0
neos-1442119	138852	10800.0546	0.0	784092	10800.0825	0.0
neos-1442657	186297	10800.1540	0.0	1795612	10800.6899	0.0
neos-1616732	199889	10800.0371	0.4	7715611	10801.3281	0.1
neos-2629914-sudost	4636	10800.0224	0.2	33779	10800.3394	0.2
neos-2978193-inde	24003	10800.0103	0.1	340787	10810.8394	0.0
neos-2978205-isar	5858	10800.5274	0.1	82470	11070.4134	0.0
neos-3072252-nete	175918	10800.0336	0.2	9360479	10800.0357	0.1
neos-3135526-osun	137402	10800.0011	1.0	9580774	10800.0632	inf

Continued on next page

Table H.1.: Full table of results for the benchmark test

	MILP-PBB 1			Gurobi		
Inst.	Nodecount	Time	Gap	Nodecount	Time	Gap
neos-3209462-rhin	442	10800.0674	1.0	1810	10804.4623	1.0
neos-3372571-onahau	1665	10800.0185	0.3	3406	10800.6498	0.1
neos-3610040-iskar	180330	10800.0813	0.1	18407	30.5564	0.0
neos-3610051-istra	154490	10800.0656	0.4	9921	58.7567	0.0
neos-3610173-itata	162830	10800.1167	0.4	9469	49.0976	0.0
neos-3611447-jijia	178235	10800.1690	0.3	11555	29.8659	0.0
neos-3611689-kaihu	177967	10800.0574	0.4	38713	77.3105	0.0
neos-3660371-kurow	39196	10800.1058	0.9	135534	4048.0624	0.0
neos-3665875-lesum	60231	10800.0010	1.0	250618	10800.4439	0.7
neos-3754480-nidda	184330	10800.0878	88.8	5474711	10800.0210	12.0
neos-4321076-ruwer	63	10800.5468	0.9	343	11585.9150	inf
neos-4333596-skien	152432	10800.0853	0.0	2446543	9727.0327	0.0
neos-480878	143829	10800.0304	0.0	13909	194.3940	0.0
neos-506422	63340	10800.0014	1.0	674	28.8153	0.0
neos-5076235-embley	6469	10800.1780	0.1	5792	10827.2823	0.2
neos-5079731-flyers	3840	10800.0478	0.2	6379	10829.0562	0.2
neos-5093327-huahum	4423	10800.2125	0.5	6955	10817.8444	0.3
neos-5100895-inster	5204	10800.4735	0.4	6834	10811.5305	0.2
neos-5102383-irwell	3518	10800.0394	0.3	4919	10836.3372	0.2
neos-5140963-mincio	210403	10800.0005	0.4	4574068	2900.5172	0.0
neos-5188808-nattai	7316	10800.1688	1.0	12008	4474.0330	0.0
neos-5192052-neckar	19	0.0535	0.0	9	0.0424	0.0
neos-5223573-tarwin	1	11286.8339	1.0	1	10811.4506	inf
neos-5251015-ogosta	1	11152.9887	1.0	1	10812.1668	inf
neos-5273874-yomtsa	1	10802.7038	inf	1	10809.5039	inf
neos-619167	78752	10800.0065	0.5	3894386	10800.1694	inf
neos-807639	130709	10800.0006	0.3	7379	112.4562	0.0
neos-848198	28062	10800.0026	0.3	687295	10801.2001	0.2
neos15	197222	10800.0144	0.8	20317568	10800.6414	0.5
neos17	164012	10800.0230	1.0	48002	174.8254	0.0
neos22	95579	10800.0395	0.9	1547871	10800.3955	0.1
neos5	234775	10800.0653	0.1	2906800	2127.3819	0.0
neos788725	234547	10800.0479	1.3	4030719	5269.0532	inf
neos858960	96721	2574.5286	1.0	3243795	5682.7336	inf
newdano	104400	10800.0204	0.8	1866388	6385.7400	0.0
nexp-50-20-1-1	209222	10800.0058	0.8	8719556	10800.0661	0.3
noswot	315792	10800.0076	0.7	498339	181.6003	0.0

Continued on next page

Table H.1.: Full table of results for the benchmark test

	MILP-PBB 1			Gurobi		
Inst.	Nodecount	Time	Gap	Nodecount	Time	Gap
ns2017839	149	2503.1319	0.0	13	130.7285	0.0
nsa	155899	10800.1330	0.3	1185854	10800.0480	0.0
osorio-cta	1	1.0746	0.0	179	14.4412	0.0
p0033	15415	147.3144	0.2	208	0.0581	0.0
p0040	135	0.1110	0.0	42	0.0077	0.0
p0201	5551	49.6902	0.0	738	1.0878	0.0
p0282	305562	10800.0362	0.6	202	0.0929	0.0
p0291	400796	10800.0071	0.8	59	0.0644	0.0
p0548	335807	10800.0213	1.0	1495	0.7414	0.0
p2756	243624	10800.0176	0.6	8117224	10800.3189	0.0
p2m2p1m1p0n100	250436	10800.0960	0.0	186637208	10800.0121	inf
p6000	63746	10800.0363	0.0	4460	48.3530	0.0
p6b	208047	10800.0295	3.2	3144535	10800.5097	2.3
p80x400b	236232	10800.0136	0.8	18054111	10800.2170	0.6
pb-market-split8-70-4	1	10800.0064	1.0	31659749	10800.0209	inf
pg	153132	10800.0441	0.4	6598829	10800.3773	0.3
pg5_34	153181	10800.0408	0.3	35873	577.8586	0.0
pigeon-08	212200	10800.0542	0.1	53476	61.8097	0.0
pigeon-10	195854	10800.0249	0.1	5201698	7586.2295	0.0
pigeon-11	203256	10800.0679	0.1	6009347	10800.4312	0.1
pipex	4445	7.7503	0.0	1318	0.2030	0.0
pk1	233481	10800.0142	1.0	226159	146.7137	0.0
pp08a	283030	10800.0224	0.7	33816171	10800.6341	0.1
pp08aCUTS	263406	10800.0311	0.3	1154039	874.9441	0.0
probportfolio	1	10800.0444	1.0	3167087	10800.0510	inf
prod1	200989	10800.0266	1.7	89081	80.8488	0.0
prod2	212463	10800.0888	4.4	393425	593.9626	0.0
qiu	135252	10800.0359	6.0	9671	163.7494	0.0
r50x360	230947	10800.0133	0.7	12756696	10800.0805	0.4
ran12x21	229438	10800.1168	0.3	1468587	1209.1800	0.0
ran13x13	226141	10800.0323	0.3	301760	192.2788	0.0
ran14x18	238475	10800.0523	0.3	15144004	10800.3489	0.1
ran14x18-disj-8	214916	10800.1172	0.3	2647123	10800.0587	0.0
ran16x16	237368	10800.0027	0.4	13051298	10482.5728	0.0
rd-rplusc-21	14336	10800.0210	1.0	25347	10801.4564	1.0
rentacar	267	76.5495	0.1	16	2.0770	0.0
rgn	5943	18.9108	0.0	2102	0.2852	0.0

Continued on next page

Table H.1.: Full table of results for the benchmark test

	MILP-PBB 1			Gurobi		
Inst.	Nodecount	Time	Gap	Nodecount	Time	Gap
rlp1	203270	10800.0277	0.4	40925616	10800.0259	0.1
rmatr100-p10	7583	2294.8982	0.0	1133	105.9677	0.0
rmatr100-p5	441	460.5619	0.0	993	237.9055	0.0
rmatr200-p10	1456	10800.0088	0.2	3442	10826.8423	0.4
rmatr200-p20	5577	10800.0051	0.2	13010	10818.3265	0.1
rmatr200-p5	201	10800.0086	0.3	1351	10827.8415	0.4
sample2	481	0.5963	0.1	89	0.0233	0.0
sentoy	4803	8.8051	0.0	112	0.0434	0.0
set1al	341672	10800.0425	0.7	40911125	10800.1220	0.2
set1ch	246933	10800.0200	0.6	28845042	10800.1036	0.2
set1cl	341645	10800.0139	0.9	42038578	10800.0376	0.5
seymour1	8784	10800.0020	0.0	9796	1979.9310	0.0
snip10x10-35r1budget17	349	10800.1876	0.7	747	10802.6392	0.6
sp150x300d	213686	10800.0429	1.0	17182426	10800.0427	0.3
stein15	273	0.2171	0.0	83	0.0224	0.0
stein15inf	367	24.6348	1.0	96	0.0429	inf
stein27	9333	23.1253	0.0	3663	0.5578	0.0
stein45	153407	4408.0642	0.0	55082	27.2081	0.0
stein45inf	244474	10800.0553	0.9	934	0.5987	inf
stein9	47	0.1549	0.0	23	0.0111	0.0
stein9inf	14	0.0893	1.0	36	0.0068	inf
sts405	106852	10800.0109	0.6	2623	10801.9358	0.6
supportcase14	147	0.8975	0.0	69	0.1508	0.0
supportcase16	245	1.5728	0.0	34	0.1633	0.0
supportcase20	154810	10800.0116	1.0	937856	10800.0561	0.9
supportcase26	199912	10800.0577	0.0	2481990	4092.8664	0.0
supportcase43	4	10800.0891	1.0	1	10808.3003	inf
tr12-30	185449	10800.0031	0.9	13507339	10800.1096	0.8
uct-subprob	156979	10800.0582	0.3	797786	10800.1632	0.1
v150d30-2hopcds	19255	10800.0051	0.4	48094	10800.1231	0.2
van	10380	10800.0284	0.7	14321	10800.8984	0.5
vpm1	315839	10800.0108	0.3	42432	15.3452	0.0
vpm2	270298	10800.0319	0.4	81683	53.6303	0.0
zib54-UUE	121090	10800.0259	0.7	659841	10801.9409	0.2

Bibliography

Achterberg, T., T. Koch, and A. Martin (2005). "Branching rules revisited". In: *Operations Research Letters* 33.1, pp. 42–54. DOI: 10.1016/j.orl.2004.04.002.

— (2006). "MIPLIB 2003". In: *Operations Research Letters* 34.4, pp. 361–372. DOI: https://doi.org/10.1016/j.orl.2005.07.009. URL: https://www.sciencedirect.com/science/article/pii/S0167637705000982.

Beck, A. (2017). *First-order methods in optimization.* Vol. 25. SIAM. DOI: 10.1137/1.9781611974997.

Benichou, M., J. M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, and O. Vincent (1971). "Experiments in mixed-integer linear programming". In: *Mathematical Programming* 1.1, pp. 76–94. DOI: 10.1007/BF01584074.

Bixby, R. E., E. A. Boyd, and R. R. Indovina (1992). "MIPLIB: A test set of mixed integer programming problems". In: *Siam News* 25.2, p. 16.

Bixby, R. E., S. Ceria, C. M. McZeal, and M. W. Savelsbergh (1998). *An updated mixed integer programming library: MIPLIB 3.0.* Tech. rep.

Chandrasekaran, R., S. N. Kabadi, and R. Sridhar (1998). "Integer Solution for Linear Complementarity Problem". In: *Mathematics of Operations Research* 23.2, pp. 390–402. DOI: 10.1287/moor.23.2.390.

Cornuéjols, G (2008). "Valid inequalities for mixed integer linear programs". In: *Mathematical Programming* 112, pp. 3–44. DOI: 10.1007/s10107-006-0086-0.

Cottle, R. W., J.-S. Pang, and R. E. Stone (2009). *The Linear Complementarity Problem.* Society for Industrial and Applied Mathematics. DOI: 10.1137/1.9780898719000.

Cunningham, W. H. and J. F. Geelen (1998). "Integral Solutions of Linear Complementarity Problems". In: *Mathematics of Operations Research* 23.1, pp. 61–68. DOI: 10.1287/moor.23.1.61.

De Santis, M., S. Lucidi, and F. Rinaldi (2013). "A new class of functions for measuring solution integrality in the Feasibility Pump approach". In: *SIAM Journal on Optimization* 23.3, pp. 1575–1606. DOI: 10.1137/110855351.

De Santis, M. and F. Rinaldi (2012). "Continuous Reformulations for Zero–One Programming Problems". In: *Journal of Optimization Theory and Applications* 153, pp. 75–84.

Dolan, E. D. and J. J. Moré (2002). "Benchmarking optimization software with performance profiles". In: *Mathematical programming* 91.2, pp. 201–213. DOI: 10.1007/s101070100263.

Dubey, D. and S. K. Neogy (2018). "Total dual integrality and integral solutions of the linear complementarity problem". In: *Linear Algebra and its Applications* 557, pp. 359–374. DOI: 10.1016/j.laa.2018.08.004.

Fomeni, F. D., S. A. Gabriel, and M. F. Anjos (2019a). "An RLT approach for solving the binary-constrained mixed linear complementarity problem". In: *Computers & Operations Research* 110, pp. 48–59. DOI: 10.1016/j.cor.2019.05.008.

— (2019b). "Applications of logic constrained equilibria to traffic networks and to power systems with storage". In: *Journal of the Operational Research Society* 70.2, pp. 310–325. DOI: 10.1080/01605682.2018.1438761.

Gabriel, S. A. (2017). "Solving discretely constrained mixed complementarity problems using a median function". In: *Optimization and Engineering* 18.3, pp. 631–658. DOI: 10.1007/s11067-012-9182-2.

Gabriel, S. A., A. J. Conejo, C. Ruiz, and S. Siddiqui (2013a). "Solving discretely constrained, mixed linear complementarity problems with applications in energy". In: *Computers & Operations Research* 40.5, pp. 1339–1350. DOI: 10.1016/j.cor.2012.10.017.

Gabriel, S. A., M. Leal, and M. Schmidt (2021). "Solving Binary-Constrained Mixed Complementarity Problems Using Continuous Reformulations". In: *Computers & Operations Research* 131. Online first. DOI: 10.1016/j.cor.2020.105208.

Gabriel, S. A., S. A. Siddiqui, A. J. Conejo, and C. Ruiz (2013b). "Solving discretely-constrained Nash–Cournot games with an application to power markets". In: *Networks and Spatial Economics* 13.3, pp. 307–326. DOI: 10.1007/s11067-012-9182-2.

Giannessi, F. and F. Tardella (1998). "Connections between nonlinear programming and discrete optimization". In: *Handbook of combinatorial optimization.* Springer, pp. 149–188. DOI: 10.1007/978-1-4613-0303-9_3.

Gleixner, A., G. Hendel, G. Gamrath, T. Achterberg, M. Bastubbe, T. Berthold, P. Christophel, K. Jarck, T. Koch, J. Linderoth, et al. (2021). "MIPLIB 2017: data-driven compilation of the 6th mixed-integer programming library". In: *Mathematical Programming Computation* 13.3, pp. 443–490.

Koch, T., T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R. E. Bixby, E. Danna, G. Gamrath, A. M. Gleixner, S. Heinz, et al. (2011). "MIPLIB 2010". In: *Mathematical Programming Computation* 3.2, pp. 103–163.

Kuhn, H. (1976). "Nonlinear Programming: A Historical View". In: *Nonlinear Programming: Proceedings of the SIAM-AMS Symposia* 9, pp. 1–26.

Land, A. and A. Doig (1960). "An Automatic Method of Solving Discrete Programming Problems". In: vol. 28. 3, pp. 497–520. DOI: 10.2307/1910129.

Lemke, C. E. and J. J. T. Howson (1962). "Equilibrium Points of Bimatrix Games". In: *Journal of The Society for Industrial and Applied Mathematics* 12, pp. 413–423.

Liberti, L., P. Milano, P Zza, and L Vinci (Mar. 2006). "Introduction to global optimization". In.

Lucidi, S. and F. Rinaldi (2010). "Exact penalty functions for nonlinear integer programming problems". In: *Journal of optimization theory and applications* 145.3, pp. 479–488. DOI: 10.1007/s10957-010-9700-7.

Nocedal, J. and S. J. Wright (1999). *Numerical Optimization.* Springer.

Pardalos, P. M. and A. Nagurney (1990). "The integer linear complementarity problem". In: *International Journal of Computer Mathematics* 31.3-4, pp. 205–214. DOI: 10.1080/00207169008803803.

Rinaldi, F. (2009). "New results on the equivalence between zero-one programming and continuous concave programming". In: *Optimization Letters* 3.3, pp. 377–386. DOI: 10.1007/s11590-009-0117-x.

Santis, M. D., S. de Vries, M. Schmidt, and L. Winkel (2022). "A Penalty Branch-and-Bound Method for Mixed Binary Linear Complementarity Problems". In: *INFORMS Journal on Computing*.

Sumita, H., N. Kakimura, and K. Makino (2018). "Total dual integrality of the linear complementarity problem". In: *Annals of Operations Research* 274.1–2. DOI: 10.1007/s10479-018-2926-8.

Weinhold, R. and S. A. Gabriel (2020). "Discretely constrained mixed complementary problems: Application and analysis of a stylised electricity market". In: *Journal of the Operational Research Society* 71.2, pp. 237–249. DOI: 10.1080/01605682.2018.1561163.

Wojtaszek, D. T. and J. W. Chinneck (2010). "Faster MIP solutions via new node selection rules". In: *Computers & Operations Research* 37.9, pp. 1544–1556. DOI: 10.1016/j.cor.2009.11.011.

Zhu, W. X. (2003). "Penalty parameter for linearly constrained 0–1 quadratic programming". In: *Journal of Optimization Theory and Applications* 116.1, pp. 229–239. DOI: 10.1023/A:1022174505886.