A COMPUTATIONAL STUDY FOR PIECEWISE LINEAR RELAXATIONS OF MIXED-INTEGER NONLINEAR PROGRAMS

KRISTIN BRAUN[1,2,*], ROBERT BURLACU[1]

[1] *Fraunhofer Institute for Integrated Circuits IIS, Nordostpark 93, D-90411 Nürnberg, Germany*

[2] *Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Discrete Optimization, Cauerstr. 11, D-91058 Erlangen, Germany*

[*] *Corresponding author,* kristin.braun@fau.de

Abstract. Solving mixed-integer nonlinear problems by means of piecewise linear relaxations can be a reasonable alternative to the commonly used spatial branch-and-bound. These relaxations have been modeled by various mixed-integer models in recent decades. The idea is to exploit the availability of mature solvers for mixed-integer problems. In this work, we compare different reformulations in terms of behavior and runtime to determine which method to apply in practice. To this end, we implement eight different mixed-integer representations for piecewise linear relaxations and evaluate them on a benchmark set from the MINLPLIB consisting of over 300 instances. We utilize existing expression trees to reformulate all nonlinearities to one-dimensional functions and afterwards compute a set of interpolation breakpoints for each function based on a given maximum error per segment. Our analysis includes a comprehensive comparison of the number of problems solved, runtimes, and optimality gaps. Overall, the classical incremental method of Markowitz and Manne 1957 has the best performance, leading to a general recommendation of this method for solving nonlinear problems by piecewise linear relaxations.

Area of Review: Optimization
Keywords: Piecewise linear relaxations, mixed-integer nonlinear programming, mixed-integer programming, discrete optimization

1. Introduction

To this day, general MINLPs remain very difficult to solve. Spatial branch-and-bound is still at the heart of most state-of-the-art solvers. However, over the last two decades, several methods have been presented that treat non-convex MINLPs by piecewise convex relaxations without direct branching of continuous variables, see for example Martin et al. 2006; Geißler et al. 2012; Morsi 2013; Lundell et al. 2013; Gugat et al. 2018; Burlacu et al. 2019; Aigner et al. 2023; Link and Volkwein 2023; Beach et al. 2022. While these approaches are sometimes quite different, they all need to tackle the following two problems: The construction of tight relaxations of the nonlinear functions and the incorporation of these relaxations into a mixed-integer linear program (MIP) or convex nonlinear program (NLP).

One approach to obtain such relaxations is to compute an optimal linearization of a nonlinear function in terms of the number of breakpoints and an a priori given accuracy as in Rebennack and Kallrath 2015b; Rebennack and Kallrath 2015a and Rebennack and Krasko 2020. This is supplemented by the construction of optimal polynomial relaxations of one-dimensional functions in Morsi 2013. Specific approximation techniques for general nonlinear functions with dimensions smaller than three are proposed in Misener and Floudas 2010. The major problem with all these methods is the curse of dimensionality, since the number of simplices in the approximation grows exponentially with the function's dimension. In this regard, we refer to the method of Rovatti et al. 2014, which avoids this problem on the basis that the piecewise linear approximation does not have to interpolate the original function at the vertices of the triangulation.

In the literature, a variety of different approaches exists for modeling PWL functions as a MIP. This article covers all major formulations: the disaggregated (as in Croxton et al. 2003; Sherali 2001) and aggregated convex combination models (as in Lee and Wilson 2001; Padberg 2000) as well as their logarithmic variants presented by Vielma et al. 2010, the classical incremental method of Markowitz and Manne 1957, and the multiple choice model as in Balakrishnan and Graves 1989 and again Croxton et al. 2003, all of which have been known for some time. Moreover, we also consider the very recently introduced binary and integer Zig-Zag formulations, which are also logarithmic versions of the convex combination models; see Huchette and Vielma 2022. For more details on the theoretical foundations of MIP models for PWL functions and general mixed-integer formulation techniques, we refer to the extensive survey by Vielma 2015.

To the best of our knowledge, no comprehensive study has yet been conducted that demonstrates the performance of these MIP models on a wide range of MINLP problems. In most cases, some parts of the mentioned MIP models are compared only on some instances of very specific problems; see for instance the works by Correa-Posada and Sánchez-Martín 2014 and Hasan and Karimi 2010. Therefore, it becomes difficult to decide in general, which MIP modeling is preferable, or if there is such a best modeling at all. This paper aims at filling exactly this remaining gap. To this end, we perform an extensive computational study on over 300 instances from the MINLPLIB. Using expression trees and the results from Bärmann et al. 2022, we reformulate all MINLP instances to equivalent models that consist of only one-dimensional nonlinear functions. Based on these reformulations, we compare various MIP relaxations corresponding to the different PWL models. Overall, our study shows that the incremental method, although the oldest approach of all, generally performs best when MINLPs are solved by PWL relaxations that are modeled as MIPs, while the very recently introduced logarithmic integer Zig-Zag model by Huchette and Vielma 2022 is a reasonable alternative for very high-accuracy PWL relaxations.

This article is structured as follows. We introduce all necessary definitions and basic ideas on how to solve MINLPs by PWL relaxations in Section 2.

Section 3 provides a detailed overview of the various MIP models for PWL relaxations. After describing our framework in Section 4, we present our comprehensive computational study in Section 5 that illustrate the practicability of the various MIP models for PWL relaxations. We discuss the numerical results in Section 6 and conclude this work in Section 7.

2. Preliminaries

We define a MINLP as an optimization problem of the following type:

$$\begin{aligned} \min_x \quad & c^\top x \\ \text{s.t.} \quad & Ax \le b, \\ & f_i(x) \le 0 \qquad \text{for all } i \in \{1, \dots, k\}, \\ & l \le x \le u, \\ & x \in \mathbb{R}^q \times \mathbb{Z}^p, \end{aligned} \tag{P}$$

where k, q, $p \in \mathbb{N}$. Initially, $Ax \le b$ denotes the linear constraints, while the continuous nonlinear real-valued functions $f_i \colon \mathbb{R}^{q+p} \to \mathbb{R}$ for $i = 1, \dots, k$ describe the nonlinear constraints. The variables x are bounded from below and above by $l, u \in \mathbb{R}^{q+p}$. Additionally, we denote by $\mathcal{F}$ the set of all nonlinear functions $f_i(x)$ and by $D_f \subset \mathbb{R}^{q+p}$ the domain of a nonlinear function $f \in \mathcal{F}$. Equality constraints, i.e., constraints of type $f_i(x) = 0$, are inherently contained in the description (P) by simply adding $f_i(x) \le 0$ and $-f_i(x) \le 0$. Please note that we are not restricted to a linear objective function $c^\top x$, since we can include any nonlinear objective function $f \colon \mathbb{R}^{q+p} \to \mathbb{R}$ by substituting $f(x)$ with a variable $y \in \mathbb{R}$ and adding $f(x) \le y$ as a constraint to the MINLP problem. As $\max\, c^\top x = -\min\, -c^\top x$, any maximization problem can be transformed to a minimization problem. Therefore, (P) serves as a comprehensive formal representation of a MINLP problem.

Since each variable in (P) has lower and upper bounds, the domain D_f is a d-dimensional box with $d \le q + p$ and its edges parallel to the coordinate axes. Thus, it is a compact set. For low dimensions d, the literature shows that using a piecewise linear (PWL) relaxation for $f \in \mathcal{F}$ can be a viable alternative to spatial branching. The most common approach is to first compute a PWL approximation of f by interpolating the function on a d-dimensional simplex, and subsequently adding the corresponding approximation error to the interpolation. To this end, a triangulation of D_f has to be constructed first. Since the triangulation of a d-dimensional box grows exponentially with the dimension d, it follows directly why this method is only useful for low dimensions d.

For a wide range of practically interesting MINLPs, however, the nonlinear functions are factorable, i.e., they can be represented by a recursive combination of elementary operators contained in the set

$$\mathcal{E} = \{+, \times, /, \hat{}, \sin, \cos, \exp, \log, |\cdot|\}; \tag{1}$$

as described by Belotti et al. 2010. The excessive use of this leads to an equivalent formulation of the MINLP problem that contains only univariate nonlinearities derived from $\mathcal{E}$ and, if necessary, the coupling bivariate function $f(x_1, x_2) = x_1 x_2$.

Following the results of Bärmann et al. 2022, we can further reformulate $f(x_1, x_2)$ to an one-dimensional representation via

$$f(x_1, x_2) = \frac{1}{2}\left(p^2 - x_1^2 - x_2^2\right) \tag{2}$$

and

$$p = x_1 + x_2 \tag{3}$$

using a new variable p. The authors show that in practice it is more favorable to use (2) and (3) instead of a bivariate product when dealing with PWL approximations. This reformulation can be further strengthened by adapting the famous McCormick relaxations from McCormick 1976:

$$\frac{1}{2}\left(p^2 - x_1^2 - x_2^2\right) \geq x_1^- \cdot x_2 + x_1 \cdot x_2^- - x_1^- \cdot x_2^-, \tag{4a}$$

$$\frac{1}{2}\left(p^2 - x_1^2 - x_2^2\right) \geq x_1^+ \cdot x_2 + x_1 \cdot x_2^+ - x_1^+ \cdot x_2^+, \tag{4b}$$

$$\frac{1}{2}\left(p^2 - x_1^2 - x_2^2\right) \leq x_1^+ \cdot x_2 + x_1 \cdot x_2^- - x_1^+ \cdot x_2^-, \tag{4c}$$

$$\frac{1}{2}\left(p^2 - x_1^2 - x_2^2\right) \leq x_1^- \cdot x_2 + x_1 \cdot x_2^+ - x_1^- \cdot x_2^+, \tag{4d}$$

where x_1^-, x_1^+, x_2^-, and x_2^+ are the lower and upper bounds of x_1 and x_2, respectively. This facilitates an equivalent reformulation of the MINLP that consists solely of one-dimensional nonlinearities. Hence, in the following, we consider only one-dimensional MIP models for PWL relaxations of nonlinear functions.

Please note that for tackling a MINLP by PWL relaxations an adaptive refinement of the PWL relaxations is usually crucial for the performance of the approach. In this article, we omit this algorithmic overhead since we are primarily interested in comparing various MIP models for PWL relaxations with different approximation errors.

3. One-dimensional MIP models for piecewise linear relaxations

In this chapter, we describe in more detail the MIP representations of the piecewise linear models that we use in this work. For all representations, we show the MIP formulation as well as a visual representation.

3.1. Disaggregated convex combination model.

The first model that we discuss is the disaggregated convex combination model. Here, each feasible point $(x, \bar{f}(x))$ is represented as a convex combination of its two neighboring breakpoints. Assuming that x lies in the i-th segment, i.e., $\bar{x}_i \leq x \leq \bar{x}_{i+1}$ holds, then x can be represented by the equality

$$x = \lambda_i^1 \bar{x}_i + \lambda_i^2 \bar{x}_{i+1} \tag{5}$$

with $\lambda_i^1 + \lambda_i^2 = 1$ and $\lambda_i^1, \lambda_i^2 \geq 0$. Analogously, $\bar{f}(x)$ is then given by

$$\bar{f}(x) = \lambda_i^1 f(\bar{x}_i) + \lambda_i^2 f(\bar{x}_{i+1}) \tag{6}$$

using the same two variables λ_i^1, λ_i^2.

Since a piecewise linear representation consists of multiple segments, we introduce a binary variable $y_i \in \{0, 1\}$ for each segment $i \in [n]$. Exactly one of these variables must be nonzero, indicating that x lies in segment i. The resulting model consists of n binary and $2n$ continuous variables. We describe all constraints in (Disag). Further, Figure 1 depicts a visual representation.

Model 1. Disaggregated convex combination model

$$\begin{aligned} \sum_{i=1}^{n}\left(\lambda_i^1 \bar{x}_{i-1} + \lambda_i^2 \bar{x}_i\right) &= x, \\ \sum_{i=1}^{n}\left(\lambda_i^1 f(\bar{x}_{i-1}) + \lambda_i^2 f(\bar{x}_i)\right) &= z, \\ \lambda_i^1 + \lambda_i^2 &= y_i, && i \in \{1, \dots, n\}, \\ \sum_{i=1}^{n} y_i &= 1, \\ \lambda_i^1, \lambda_i^2 &\geq 0, && i \in \{1, \dots, n\}, \\ y_i &\in \{0,1\}, && i \in \{1, \dots, n\}. \end{aligned} \tag{Disag}$$

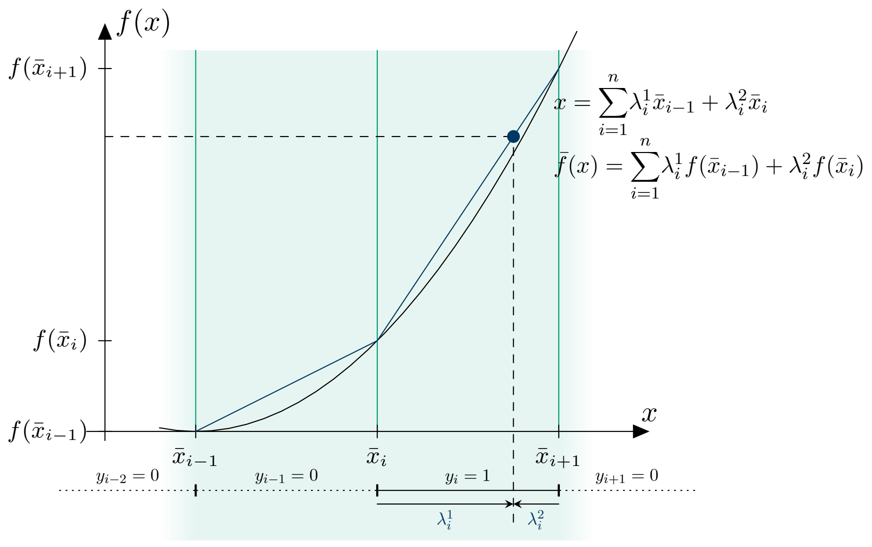


FIGURE 1. Representation of (Disag).

Given that it is possible to encode an n-digit number using $\lceil \log_2 n \rceil$ binary variables, the idea to create a binary representation of the segments 1 to n is obvious. To this end, we introduce binary variables y_l with $l \in \{1, ..., \lceil \log_2 n \rceil\}$ and a binary encoding $B : [n] \to \{0,1\}^{\lceil \log_2 n \rceil}$. The binary encoding is not fixed; in our case, we simply use the usual conversion from decimal to binary system. Using this encoding, we define a branching scheme

$$\mathscr{P}^0(B, l) := \{i \in [n] \mid B(i)_l = 0\} \tag{7}$$

and

$$\mathscr{P}^+(B, l) := \{i \in [n] \mid B(i)_l = 1\}. \tag{8}$$

Now, as an arbitrary number of binary variables can be nonzero, the representation has to be changed to ensure that $\lambda_i^1 + \lambda_i^2 = 1$ holds for exactly one $i \in [n]$ and $\lambda_j^1 + \lambda_j^2 = 0$ holds for all $j \in [n]$ with $i \neq j$. These adjustments result in (LogDisag) that uses the same number of continuous variables as in (Disag), but now only $\lceil \log_2 n \rceil$ binary variables.

Model 2. Logarithmic disaggregated convex combination model

$$\begin{aligned}
\sum_{i=1}^{n} \left(\lambda_i^1 \bar{x}_{i-1} + \lambda_i^2 \bar{x}_i\right) &= x, \\
\sum_{i=1}^{n} \left(\lambda_i^1 f(\bar{x}_{i-1}) + \lambda_i^2 f(\bar{x}_i)\right) &= z, \\
\sum_{i=1}^{n} \lambda_i^1 + \lambda_i^2 &= 1, \\
\sum_{i \in \mathscr{P}^+(B,l)} \lambda_i^1 + \lambda_i^2 &\leq y_l, && l \in \{1, \dots, \lceil \log_2 n \rceil\}, \\
\sum_{i \in \mathscr{P}^0(B,l)} \lambda_i^1 + \lambda_i^2 &\leq 1 - y_l, && l \in \{1, \dots, \lceil \log_2 n \rceil\}, \\
\lambda_i^1, \lambda_i^2 &\geq 0, && i \in \{1, \dots, n\}, \\
y_l &\in \{0, 1\}, && l \in \{1, \dots, \lceil \log_2 n \rceil\}.
\end{aligned} \tag{LogDisag}$$

3.2. **Aggregated convex combination model.** For introducing the aggregated convex combination model, we just need to adjust the previous model slightly.

In (Disag), variables λ_{i-1}^2 and λ_i^1 represent the same tuple $(\bar{x}_i, f(\bar{x}_i))$ and, thus, we can aggregate them. To this end, we introduce new continuous variables $\lambda_0, \dots, \lambda_n$. The new variable λ_0 replaces λ_1^1, λ_n replaces λ_n^2, and, for $i = 1, \dots, n-1$, λ_i replaces λ_{i-1}^2 and λ_i^1. Finally, we only need $n+1$ instead of $2n$ continuous variables. Now, a variable λ_i is allowed to be nonzero if either segment i or segment $i+1$ is used, i.e., $\lambda_i \leq y_i + y_{i+1}$. The constraints are given in (Ag), and, again, Figure 2 provides a visual representation.

Model 3. Convex combination model

$$\begin{aligned}
\sum_{i=0}^{n} \lambda_i \bar{x}_i &= x, \\
\sum_{i=0}^{n} \lambda_i f(\bar{x}_i) &= z, \\
\sum_{i=0}^{n} \lambda_i &= 1, \\
\lambda_0 &\leq y_1, \\
\lambda_i &\leq y_i + y_{i+1}, && i \in \{1, \dots, n-1\}, \\
\lambda_n &\leq y_n, \\
\sum_{i=1}^{n} y_i &= 1, \\
\lambda_i &\geq 0, && i \in \{0, \dots, n\}, \\
y_i &\in \{0, 1\}, && i \in \{1, \dots, n\}.
\end{aligned} \tag{Ag}$$

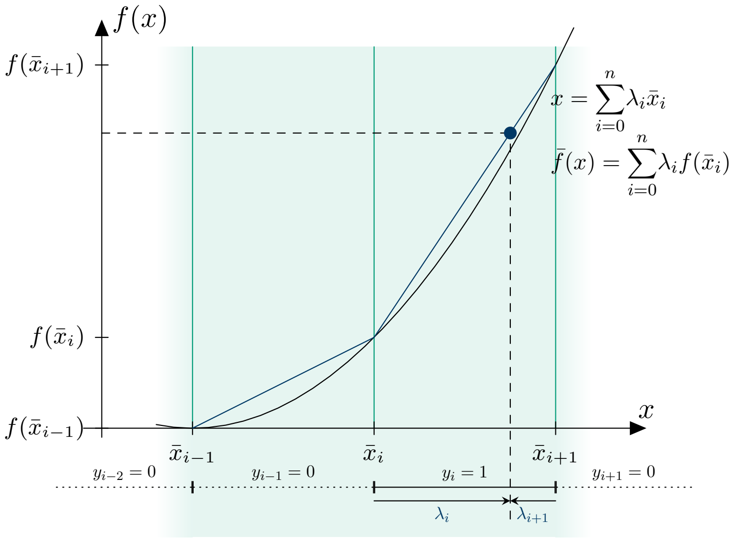


FIGURE 2. Representation of (Ag).

As before, we want to reduce the number of binary variables in our model. To this end, we introduce different approaches: On the one hand, a formulation using a binary branching scheme, and, on the other hand, two new formulations introduced by Huchette and Vielma 2022.

In (Disag) and (LogDisag), each continuous variable λ_i belongs to two different segments. The binary branching scheme of (Disag) fixes all variables λ_i^1, λ_i^2 to zero if segment i is not used. If we use the same branching scheme here in the aggregated version, this would result in fixing all continuous variables to zero and then no segment is usable anymore. Thus, we need to define a branching scheme L_s and R_s such that for all $i \in \{1, \ldots, n\}$ there exists a series $T^i = [T_1^1, \ldots, T_S^i]$ with

$$\{\bar{x}_{i-1}, \bar{x}_i\} = \bigcap_{s=1}^{S} \bar{X}_n \setminus T_s^i, \tag{9}$$

where $T_s^i \in \{L_s, R_s\}$ and $\bar{X}_n$ abbreviates the set $\{\bar{x}_0, \ldots, \bar{x}_n\}$.

We describe a branching scheme L_s, R_s for $s \in [S]$ with $S := \lceil \log_2 n \rceil$ in the following. For $n \leq 2$, i.e., $S = 1$, we set

$$L_1 := \{\bar{x}_0\} \cap \{\bar{x}_0, \ldots, \bar{x}_n\}, R_1 := \{\bar{x}_2\} \cap \{\bar{x}_0, \ldots, \bar{x}_n\}. \tag{10}$$

For any $n > 2$, the sets $L_1, \ldots, L_S$ and $R_1, \ldots, R_S$ are defined recursively by expansion based on mirroring: We assume that $\bar{L}_1, \ldots, \bar{L}_{S-1}$ and $\bar{R}_1, \ldots, \bar{R}_{S-1}$ are given for $\bar{n} = 2^{S-1}$ and define

$$\begin{aligned} L_S &:= \left\{\bar{x}_0, \ldots, \bar{x}_{2^{S-1}-1}\right\}, \\ R_S &:= \left\{\bar{x}_{2^{S-1}+1}, \ldots, \bar{x}_n\right\}, \end{aligned} \tag{11}$$

and

$$\begin{aligned} L_s &:= \left(\bar{L}_s \cup \left\{\bar{x}_{2^S - j} : \bar{x}_j \in \bar{L}_s\right\}\right) \cap \{\bar{x}_0, \ldots, \bar{x}_n\}, \\ R_s &:= \left(\bar{R}_s \cup \left\{\bar{x}_{2^S - j} : \bar{x}_j \in \bar{R}_s\right\}\right) \cap \{\bar{x}_0, \ldots, \bar{x}_n\} \end{aligned} \tag{12}$$

for all $s \in \{1, \dots, S-1\}$. Lemma 4 shows that these sets fulfill the necessary conditions.

Lemma 4. *Let a partition of $[\bar{x}_0, \bar{x}_n]$ into n segments with $n+1$ breakpoints $\bar{x}_0, \dots, \bar{x}_n$ be given. Further, let L_s, R_s be defined as above for $s \in \{1, \dots, \lceil \log_2(n) \rceil\}$. Then, for all segments $[\bar{x}_{i-1}, \bar{x}_i]$ with $1 \leq i \leq n$, there exists a series of sets $T^i := \left[T_1^i, \dots, T^i_{\lceil \log_2(n) \rceil - 1}\right]$ with $T_s^i \in \{L_s, R_s\}$ such that*

$$\bar{X}_n \setminus \{\bar{x}_{i-1}, \bar{x}_i\} = \bigcup_{s=1}^{\lceil \log_2(n) \rceil} T_s^i, \tag{13}$$

i.e., we can represent the vertices $\bar{x}_{i-1}$ and $\bar{x}_i$ of each segment by a disjunction of $\lceil \log_2(n) \rceil$ pre-defined sets.

Proof. We can assume w.l.o.g. that n is a power of two, i.e., there is some $S \in \mathbb{N}^+$ with $S = \log_2 n$ and proof the lemma via induction in the following.

For $n = 2$, i.e., $S = 1$, let $T_1^1 := R_1$ and $T_1^2 := L_1$. Then, we have

$$\bigcup_{s=1}^{S} T_s^i = \begin{cases} R_1 = \{\bar{x}_2\} = \bar{X}_2 \setminus \{\bar{x}_0, \bar{x}_1\} & \text{if } i = 1, \\ L_1 = \{\bar{x}_0\} = \bar{X}_2 \setminus \{\bar{x}_1, \bar{x}_2\} & \text{if } i = 2. \end{cases} \tag{14}$$

Now, let $n > 2$, i.e., $S > 1$: We assume that for any $i \in \{1, \dots, 2^{S-1}\}$, there is a series $\bar{T}^i := \left[\bar{T}_1^i, \dots, \bar{T}_{S-1}^i\right]$ with $\bar{T}_s^i \in \{\bar{L}_s, \bar{R}_s\}$ such that

$$\bar{X}_{2^{S-1}} \setminus \{\bar{x}_{i-1}, \bar{x}_i\} = \bigcup_{s=1}^{S-1} \bar{T}_s^i. \tag{15}$$

It further holds that for any $i \in \{1, \dots, 2^{S-1}\}$,

$$\begin{aligned} \bigcup_{s=1}^{S-1} T_s^i &= \bigcup_{s=1}^{S-1} \left(\bar{T}_s^i \cup \left\{ \bar{x}_{2^S - j} : \bar{x}_j \in \bar{T}_s^i \right\} \right) \\ &= \bigcup_{s=1}^{S-1} \bar{T}_s^i \cup \bigcup_{s=1}^{S-1} \left\{ \bar{x}_{2^S - j} : \bar{x}_j \in \bar{T}_s^i \right\} \\ &= \bigcup_{s=1}^{S-1} \bar{T}_s^i \cup \left\{ \bar{x}_{2^S - j} : \bar{x}_j \in \bigcup_{s=1}^{S-1} \bar{T}_s^i \right\}. \end{aligned} \tag{16}$$

Using (15), we can reformulate this to

$$\begin{aligned} \bigcup_{s=1}^{S-1} T_s^i &= \left(\bar{X}_{2^{S-1}} \setminus \{\bar{x}_{i-1}, \bar{x}_i\} \right) \cup \left(\left\{ \bar{x}_{2^S - j} : \bar{x}_j \in \bar{X}_{2^{S-1}} \setminus \{\bar{x}_{i-1}, \bar{x}_i\} \right\} \right) \\ &= \left(\bar{X}_{2^{S-1}} \setminus \{\bar{x}_{i-1}, \bar{x}_i\} \right) \cup \left(\{\bar{x}_{2^{S-1}}, \dots, \bar{x}_{2^S}\} \setminus \left\{ \bar{x}_{2^S - i}, \bar{x}_{2^S - (i-1)} \right\} \right) \\ &= \bar{X}_{2^S} \setminus \left\{ \bar{x}_{i-1}, \bar{x}_i, \bar{x}_{2^S - i}, \bar{x}_{2^S - (i-1)} \right\} \\ &= \bar{X}_n \setminus \left\{ \bar{x}_{i-1}, \bar{x}_i, \bar{x}_{2^S - i}, \bar{x}_{2^S - (i-1)} \right\}. \end{aligned} \tag{17}$$

Now, we need to prove that, using the new sets L_S and R_S, we can find a series of sets $T^i := [T_1^i, \dots, T_S^i]$ such that (13) holds for all $i \in \{1, \dots, n\}$.

If $i \leq 2^{S-1}$, we choose $T_s^i = \bar{T}_s^i$ for $s \in \{1, \dots, S-1\}$ and $T_S^i = R_S$. Then,

$$\begin{aligned}\bigcup_{s=1}^{S} T_s^i = T_S^i \cup \bigcup_{s=1}^{S-1} T_s^i &= R_S \cup \left(\bar{X}_n \setminus \left\{\bar{x}_{i-1}, \bar{x}_i, \bar{x}_{2^S-i}, \bar{x}_{2^S-(i-1)}\right\}\right) \\ &= \{\bar{x}_{2^{S-1}+1}, \dots, \bar{x}_n\} \cup \left(\bar{X}_n \setminus \left\{\bar{x}_{i-1}, \bar{x}_i, \bar{x}_{2^S-i}, \bar{x}_{2^S-(i-1)}\right\}\right) \\ &= \bar{X}_n \setminus \{\bar{x}_{i-1}, \bar{x}_i\}.\end{aligned} \tag{18}$$

For $i > 2^{S-1}$, the proof works similarly with $T_s^i = L_S$:

$$\begin{aligned}\bigcup_{s=1}^{S} T_s^i = T_S^i \cup \bigcup_{s=1}^{S-1} T_s^i &= L_S \cup \left(\bar{X}_n \setminus \left\{\bar{x}_{i-1}, \bar{x}_i, \bar{x}_{2^S-i}, \bar{x}_{2^S-(i-1)}\right\}\right) \\ &= \{\bar{x}_0, \dots \bar{x}_{2^{S-1}-1}\} \cup \left(\bar{X}_n \setminus \left\{\bar{x}_{i-1}, \bar{x}_i, \bar{x}_{2^S-i}, \bar{x}_{2^S-(i-1)}\right\}\right) \\ &= \bar{X}_n \setminus \left\{\bar{x}_{2^S-i}, \bar{x}_{2^S-(i-1)}\right\}.\end{aligned}$$

□

Using this branching scheme, (LogAg) gives the first logarithmic version of (Ag).

Model 5. Logarithmic branching convex combination model

$$\begin{aligned}\sum_{i=0}^{n} \lambda_i \bar{x}_i &= x, \\ \sum_{i=0}^{n} \lambda_i f(\bar{x}_i) &= z, \\ \sum_{i=0}^{n} \lambda_i &= 1, \\ \sum_{i \in L_s} \lambda_i &\leq y_s, && s \in S, \\ \sum_{i \in R_s} \lambda_i &\leq 1 - y_s, && s \in S, \\ \lambda_i &\geq 0, && i \in \{0, \dots, n\}, \\ y_s &\in \{0, 1\}, && s \in S.\end{aligned} \tag{LogAg}$$

Additionally, two new reformulations are provided by Huchette and Vielma 2022. Therein, a new encoding C^r is used. It is defined recursively as

$$\begin{aligned}C^1 &= \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \\ C^{k+1} &= \begin{pmatrix} C^k & 0^d \\ C^k + 1^d \times C_d^k & 1^d \end{pmatrix} \text{ for } k = 1, \dots, r-1,\end{aligned} \tag{19}$$

where $d = 2^k$. Further, for the sake of simplicity, they set $C_0^k = C_1^k$ and $C_{d+1}^k = C_d^k$. This encoding helps to provide the following equations:

$$\sum_{v=0}^{n} C_{v,k}^r \lambda_v \leq y_k + \sum_{l=k+1}^{r} 2^{l-k-1} y_l \leq \sum_{v=0}^{n} C_{v+1,k}^r \lambda_v, \tag{20}$$

$$\sum_{v=0}^{n} C^r_{v,k}\lambda_v \le y_k \le \sum_{v=0}^{n} C^r_{v+1,k}\lambda_v. \tag{21}$$

Using each of these two equations, one can then define two new models, a binary formulation in (BinZigZag) and an integer one in (IntZigZag). They are called Zig-Zag formulations as that describes the behavior of the single vectors. A proof for the correctness of these equations is given by Huchette and Vielma 2022.

Model 6. Binary Zig-Zag Formulation

$$\begin{aligned}
\sum_{i=0}^{n} \lambda_i \bar{x}_i &= x, \\
\sum_{i=0}^{n} \lambda_i f(\bar{x}_i) &= z, \\
\sum_{i=0}^{n} \lambda_i &= 1, \\
\sum_{v=0}^{n} C^r_{v,k}\lambda_v &\le y_k + \sum_{l=k+1}^{r} 2^{l-k-1} y_l, && k \in \{1,\dots,r\} \\
y_k + \sum_{l=k+1}^{r} 2^{l-k-1} y_l &\le \sum_{v=0}^{n} C^r_{v+1,k}\lambda_v, && k \in \{1,\dots,r\} \\
\lambda_i &\ge 0, && i \in \{0,\dots,n\}, \\
y_k &\in \{0,1\}, && k \in \{1,\dots,r\},
\end{aligned} \tag{BinZigZag}$$

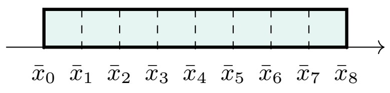
with $r = \lceil \log_2 (n-1) \rceil$.

Model 7. General Integer Zig-Zag Formulation

$$\begin{aligned}
\sum_{i=0}^{n} \lambda_i \bar{x}_i &= x, \\
\sum_{i=0}^{n} \lambda_i f(\bar{x}_i) &= z, \\
\sum_{i=0}^{n} \lambda_i &= 1, \\
\sum_{v=0}^{n} C^r_{v,k}\lambda_v &\le y_k, && k \in \{1,\dots,r\}, \\
y_k &\le \sum_{v=0}^{n} C^r_{v+1,k}\lambda_v, && k \in \{1,\dots,r\}, \\
\lambda_i &\ge 0, && i \in \{0,\dots,n\}, \\
y_k &\in \mathbb{Z}, && k \in \{1,\dots,r\},
\end{aligned} \tag{IntZigZag}$$

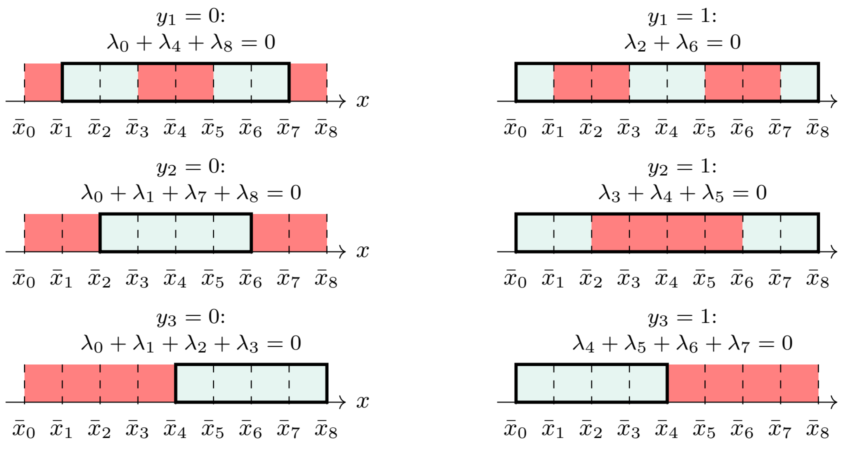
with $r = \lceil \log_2 (n-1) \rceil$.

The big difference between (LogAg) and the Zig-Zag methods (BinZigZag) and (IntZigZag) is their branching behavior. For an efficient solving of MIPs, it is important to quickly find tight dual bounds that are obtained by solving



x
$\bar{x}_0$ $\bar{x}_1$ $\bar{x}_2$ $\bar{x}_3$ $\bar{x}_4$ $\bar{x}_5$ $\bar{x}_6$ $\bar{x}_7$ $\bar{x}_8$

(3A) Initial segments



$y_1 = 0$:
$\lambda_0 + \lambda_4 + \lambda_8 = 0$
$y_1 = 1$:
$\lambda_2 + \lambda_6 = 0$
$y_2 = 0$:
$\lambda_0 + \lambda_1 + \lambda_7 + \lambda_8 = 0$
$y_2 = 1$:
$\lambda_3 + \lambda_4 + \lambda_5 = 0$
$y_3 = 0$:
$\lambda_0 + \lambda_1 + \lambda_2 + \lambda_3 = 0$
$y_3 = 1$:
$\lambda_4 + \lambda_5 + \lambda_6 + \lambda_7 = 0$
x
$\bar{x}_0$ $\bar{x}_1$ $\bar{x}_2$ $\bar{x}_3$ $\bar{x}_4$ $\bar{x}_5$ $\bar{x}_6$ $\bar{x}_7$ $\bar{x}_8$

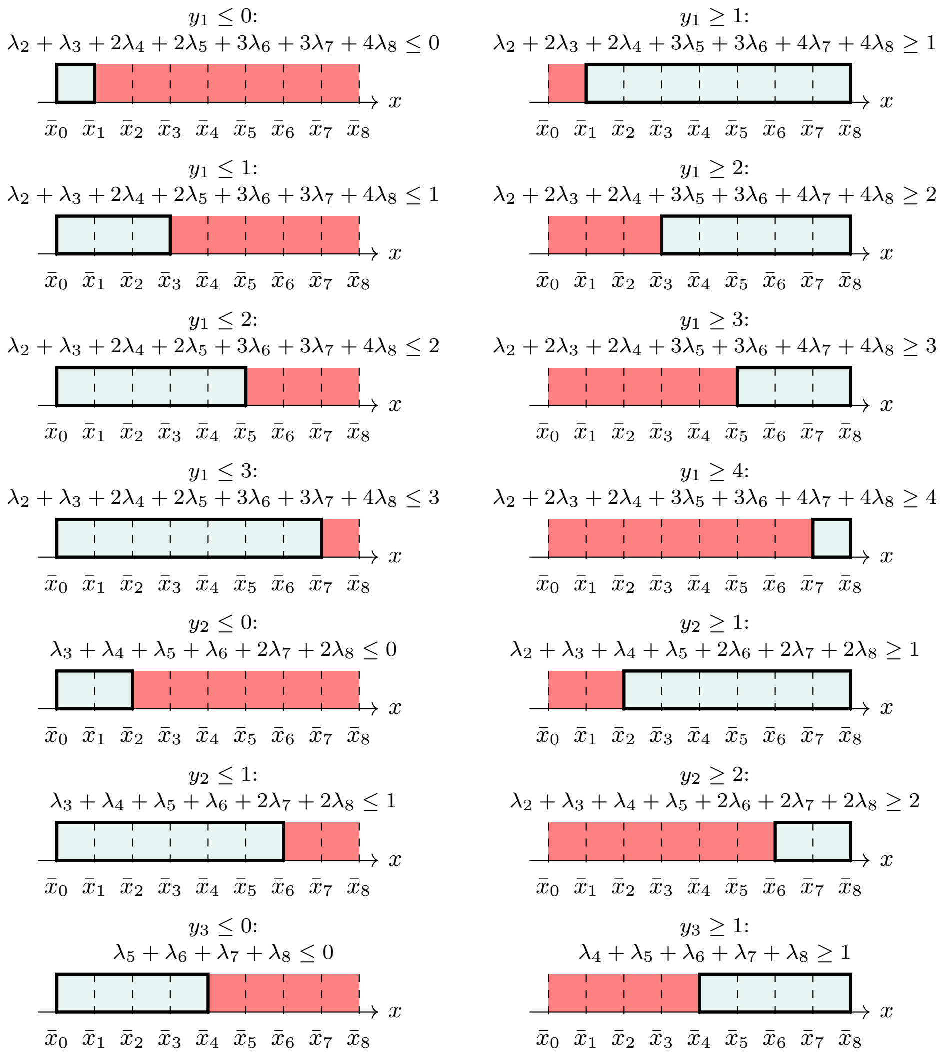
(3B) Branching for (LogAg)

LP relaxations. Each of these LP relaxation contains all feasible points, or, better said, the convex hull of all feasible points. If the LP relaxations are tight, we can find better dual bounds. In Huchette and Vielma 2022, the authors mention different metrics to evaluate the branching behavior. Besides that, we exemplarily investigate this using an example with $n = 8$ uniformly sized segments in the following.

In Figure 3a, one can see the breakpoints $\bar{x}_0, \bar{x}_1, \dots, \bar{x}_8$. These breakpoints create eight segments that are, initially, all feasible, as there was no branching up to now. Feasible segments are displayed in light blue. Further, the convex hull of the feasible set is outlined by a thick rectangle. Here, the convex hull is equivalent to the full domain.

In Figure 3b, all possible branchings for (LogAg) are shown. As we have eight segments, there are 3 binary variables y_1, y_2 and y_3 that can be either zero or one, resulting in six different branches. All possibilities are displayed where the red parts mean that segments are excluded due to the chosen branch, blue segments are still feasible. There are three cases, where the convex hull also contains infeasible regions as they lie between feasible ones. Thus, a solution of the LP relaxation can also lie inside these infeasible regions. The ZigZag reformulations try to overcome this problem.

We visualize the branching for (IntZigZag) in Figure 3c, but the behavior of (BinZigZag) is similar. In theory, there are more possible branching steps than the displayed ones, because in this reformulation the variables y_1, y_2 and y_3 are integral instead of binary. However, all other branchings would result in a combination of one infeasible branch and one branch that still contains the full domain, and, therefore, no progress at all. As one can see, all branchings lead to subdomains that are connected what means that their


$y_1 \leq 0$: $\lambda_2 + \lambda_3 + 2\lambda_4 + 2\lambda_5 + 3\lambda_6 + 3\lambda_7 + 4\lambda_8 \leq 0$
$y_1 \geq 1$: $\lambda_2 + 2\lambda_3 + 2\lambda_4 + 3\lambda_5 + 3\lambda_6 + 4\lambda_7 + 4\lambda_8 \geq 1$
$y_1 \leq 1$: $\lambda_2 + \lambda_3 + 2\lambda_4 + 2\lambda_5 + 3\lambda_6 + 3\lambda_7 + 4\lambda_8 \leq 1$
$y_1 \geq 2$: $\lambda_2 + 2\lambda_3 + 2\lambda_4 + 3\lambda_5 + 3\lambda_6 + 4\lambda_7 + 4\lambda_8 \geq 2$
$y_1 \leq 2$: $\lambda_2 + \lambda_3 + 2\lambda_4 + 2\lambda_5 + 3\lambda_6 + 3\lambda_7 + 4\lambda_8 \leq 2$
$y_1 \geq 3$: $\lambda_2 + 2\lambda_3 + 2\lambda_4 + 3\lambda_5 + 3\lambda_6 + 4\lambda_7 + 4\lambda_8 \geq 3$
$y_1 \leq 3$: $\lambda_2 + \lambda_3 + 2\lambda_4 + 2\lambda_5 + 3\lambda_6 + 3\lambda_7 + 4\lambda_8 \leq 3$
$y_1 \geq 4$: $\lambda_2 + 2\lambda_3 + 2\lambda_4 + 3\lambda_5 + 3\lambda_6 + 4\lambda_7 + 4\lambda_8 \geq 4$
$y_2 \leq 0$: $\lambda_3 + \lambda_4 + \lambda_5 + \lambda_6 + 2\lambda_7 + 2\lambda_8 \leq 0$
$y_2 \geq 1$: $\lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 + 2\lambda_6 + 2\lambda_7 + 2\lambda_8 \geq 1$
$y_2 \leq 1$: $\lambda_3 + \lambda_4 + \lambda_5 + \lambda_6 + 2\lambda_7 + 2\lambda_8 \leq 1$
$y_2 \geq 2$: $\lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 + 2\lambda_6 + 2\lambda_7 + 2\lambda_8 \geq 2$
$y_3 \leq 0$: $\lambda_5 + \lambda_6 + \lambda_7 + \lambda_8 \leq 0$
$y_3 \geq 1$: $\lambda_4 + \lambda_5 + \lambda_6 + \lambda_7 + \lambda_8 \geq 1$
x
$\bar{x}_0$ $\bar{x}_1$ $\bar{x}_2$ $\bar{x}_3$ $\bar{x}_4$ $\bar{x}_5$ $\bar{x}_6$ $\bar{x}_7$ $\bar{x}_8$

(3c) Branching for (IntZigZag)

FIGURE 3. All possible branching steps for $n = 8$ segments. Branching results in segments being feasible or infeasible, what is represented by shading: The light blue areas represent feasible regions, while the red areas represent the infeasible segments. The bold rectangle outlines the convex hull.

convex hulls are exactly the feasible set. Thus, it is not possible to have LP solutions that lie outside the feasible regions what should increase their quality and, thus, the dual bounds.

The following two equations show the inequalities we used to create the visualizations in Figures 3b and 3c. The continuous variables in (LogAg) are constrained by

$$\lambda_0 + \lambda_1 + \lambda_2 + \lambda_3 \leq y_3, \tag{22a}$$

$$\lambda_5 + \lambda_6 + \lambda_7 + \lambda_8 \leq 1 - y_3, \tag{22b}$$
$$\lambda_0 + \lambda_1 + \lambda_7 + \lambda_8 \leq y_2, \tag{22c}$$
$$\lambda_3 + \lambda_4 + \lambda_5 \leq 1 - y_2, \tag{22d}$$
$$\lambda_0 + \lambda_4 + \lambda_8 \leq y_1, \tag{22e}$$
$$\lambda_2 + \lambda_6 \leq 1 - y_1. \tag{22f}$$

When a binary variable is fixed, there is a subset of continuous variables that also need to be zero then. For example, when setting $y_1 = 0$, we obtain $\lambda_0 + \lambda_4 + \lambda_8 = 0$, i.e., $\lambda_0 = \lambda_4 = \lambda_8 = 0$ from (22e) and, thus, the feasible regions shown in the left upper plot of Figure 3b. For $y_1 = 1$, we, complementary, obtain $\lambda_2 = \lambda_6 = 0$ from (22f) what is visualized in the right upper plot.

In contrast, the inequalities in (IntZigZag) and (BinZigZag) have the form

$$\begin{aligned}&\lambda_2 + \lambda_3 + 2\lambda_4 + 2\lambda_5 + 3\lambda_6 + 3\lambda_7 + 4\lambda_8\\ &\quad \leq y_1 + y_2 + 2y_3\\ &\quad \leq \lambda_1 + \lambda_2 + 2\lambda_3 + 2\lambda_4 + 3\lambda_5 + 3\lambda_6 + 4\lambda_7 + 4\lambda_8,\end{aligned} \tag{23a}$$
$$\begin{aligned}&\lambda_3 + \lambda_4 + \lambda_5 + \lambda_6 + 2\lambda_7 + 2\lambda_8\\ &\quad \leq y_2 + y_3\\ &\quad \leq \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 + 2\lambda_6 + 2\lambda_7 + 2\lambda_8,\end{aligned} \tag{23b}$$
$$\begin{aligned}&\lambda_5 + \lambda_6 + \lambda_7 + \lambda_8\\ &\quad \leq y_3\\ &\quad \leq \lambda_4 + \lambda_5 + \lambda_6 + \lambda_7 + \lambda_8,\end{aligned} \tag{23c}$$

whereas the blue parts are the addends that are only present in (BinZigZag). It directly stands out that all inequalities only use adjacent variables.

Let us exemplarily assume that we branch on y_1, using the branches $y_1 \leq 2$ and $y_1 \geq 3$, given in the third line of Figure 3c. From (23a), we obtain

$$\lambda_1 + \lambda_2 + 2\lambda_3 + 2\lambda_4 + 3\lambda_5 + 3\lambda_6 + 4\lambda_7 + 4\lambda_8 \leq 2 \tag{24}$$

for the left branch with $y_1 \leq 2$. Using one of the variables λ_6, λ_7 or λ_8 would result in a sum greater than two, as the sum of exactly two neighboring variables λ_i and λ_{i+1} has to be one. Thus, we cannot use the three rightmost segments. In contrast,

$$3 \leq \lambda_2 + \lambda_3 + 2\lambda_4 + 2\lambda_5 + 3\lambda_6 + 3\lambda_7 + 4\lambda_8 \tag{25}$$

emerges for $y_1 \geq 3$. Here, using any variable of $\lambda_0, \lambda_1, \ldots, \lambda_5$ would yield a sum less than three with the same argument as before. Thus, we can use exactly the three rightmost segments that were cut off in the other branch.

3.3. **Incremental model.**

The incremental method was introduced by Markowitz and Manne 1957. It uses a linear number of binary variables, but a different behavior compared to the convex combination formulations. There are $n - 1$ binary variables $y_1, \ldots, y_{n-1}$ and a value $y_j = 1$ enforces that any segment i with $i \geq j$ is used. The other way around, we can say that if segment i is used, we have $y_j = 1$ for all $j < i$ and $y_j = 0$ for all $j > i$. Further, we have $y_i = 1$. The idea behind setting all leftmost binary

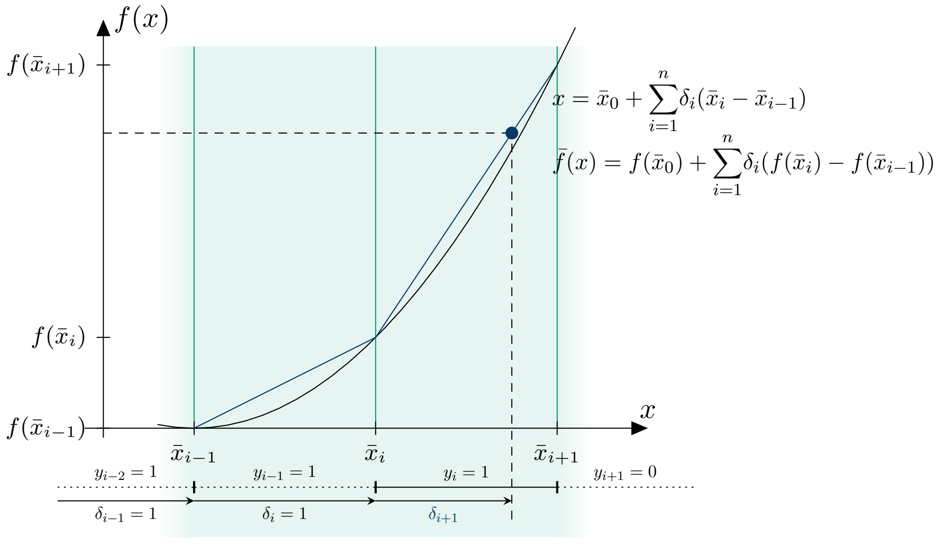


FIGURE 4. Representation of (Inc)

variables to 1 is again to solve the branching issues described for (LogAg). In this formulation, there are n continuous variables $\delta_i, \dots, \delta_n$. They are used similarly to the binary ones, i.e., $\delta_j = 1$ for all $j < i$ and $\delta_j = 0$ for all $j > i$. Moreover, the variable δ_i determines the exact position of x that is calculated by

$$x = \bar{x}_0 + \sum_{i=1}^{n} \delta_i(\bar{x}_i - \bar{x}_{i-1}). \tag{26}$$

The calculation of $f(x)$ works equivalently. The following constraint is introduced to ensure that the continuous variables behave as expected:

$$1 \geq \delta_1 \geq y_1 \geq \cdots \geq \delta_{n-1} \geq y_{n-1} \geq \delta_n \geq 0. \tag{27}$$

Due to the variable names, the incremental method is also known as the δ-method. The convex combination formulations, on the other hand, are frequently referred to as the λ-methods. Again, one can find the model in (Inc) and a visual representation in Figure 4.

Model 8. Classical Incremental Method

$$\begin{aligned} \bar{x}_0 + \sum_{i=1}^{n} \delta_i(\bar{x}_i - \bar{x}_{i-1}) &= x, \\ f(\bar{x}_0) + \sum_{i=1}^{n} \delta_i(f(\bar{x}_i) - f(\bar{x}_{i-1})) &= z, \\ \delta_1 &\leq 1, \\ \delta_{i+1} &\leq y_i, && i \in \{1, \dots, n-1\}, \\ y_i &\leq \delta_i, && i \in \{1, \dots, n-1\}, \\ \delta_n &\geq 0, \\ y_i &\in \{0, 1\}, && i \in \{1, \dots, n-1\}. \end{aligned} \tag{Inc}$$

3.4. **Multiple choice model.** Another reformulation that employs a linear number of binary variables is the multiple choice model. As in the convex combination models, each variable y_i represents one segment i, and the variables are one-hot encoded. The main idea here is that there is also exactly one nonzero continuous variable. So, for each segment i, there is a variable x_i that represents the exact value on the x-axis. For each variable x_i, we have one constraint of the form

$$y_i \bar{x}_{i-1} \leq x_i \leq y_i \bar{x}_i. \tag{28}$$

If we use segment i, we have $\bar{x}_{i-1} \leq x_i \leq \bar{x}_i$, and, thus, x_i is forced to lie exactly in the segment; otherwise, x_i is fixed to zero. Given the values of all variables x_i, we can calculate the approximation value using the parameters m_i and t_i and linear equations

$$\sum_{i=1}^{n} m_i x_i + t_i y_i. \tag{29}$$

Putting everything together, we obtain (MC) and the visual representation in Figure 5.

Model 9. Multiple Choice Model

$$\begin{aligned} \sum_{i=1}^{n} x_i &= x, \\ \sum_{i=1}^{n} (m_i x_i + t_i y_i) &= z, \\ y_i \bar{x}_{i-1} &\leq x_i, && i \in \{1, \ldots, n\}, \\ x_i &\leq y_i \bar{x}_i, && i \in \{1, \ldots, n\}, \\ \sum_{i=1}^{n} y_i &= 1, \\ y_i &\in \{0, 1\}, && i \in \{1, \ldots, n\}. \end{aligned} \tag{MC}$$

Finally, we provide an overview of the number of variables that are needed to model the MIP representations of this section in Table 1. For relaxations, we need another variable for each segment, i.e., n additional variables. This is the same for each representation

4. Implementation

After introducing the different possibilities to represent MIP relaxations of MINLPs, we go more into detail about how we implemented our computational study. For that, we firstly describe how the input instances look like before heading over to the reformulation and solving process.

4.1. **Input.** All of our input problems are in the Optimization Services Instance Language, or OSIL, format. This format employs an XML vocabulary, which provides several advantages, the most notable of which is that nonlinearities are stored in a tree-based structure. In an OSIL file, each variable, constraint, etc. is saved along with various attributes such as coefficients, a

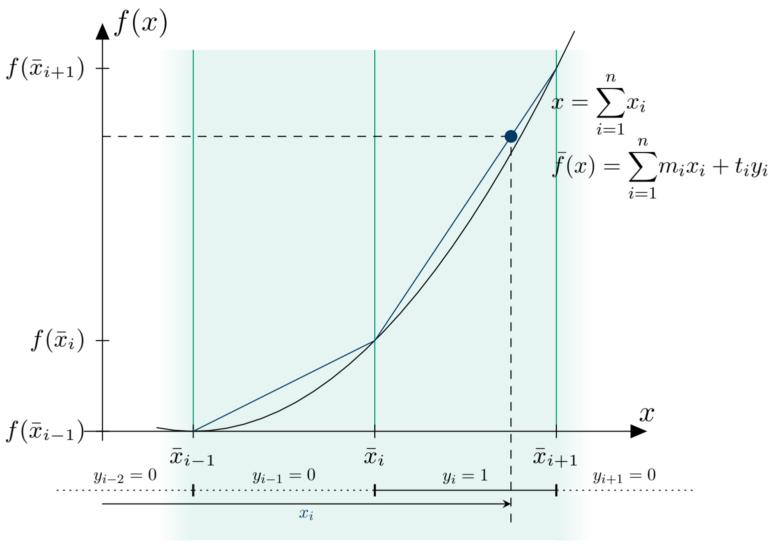


FIGURE 5. Visual representation of (MC)

TABLE 1. Sizes of all one-dimensional representations assuming that we have n segments and $n+1$ breakpoints

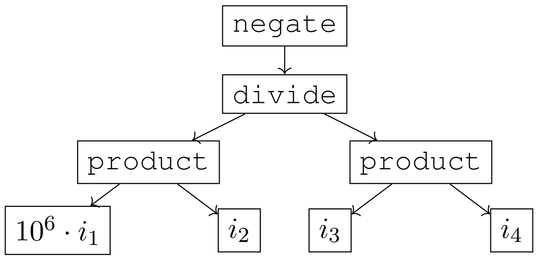
Model	Constraints	Variables		
		continuous	binary	integer
(Disag)	$n+3$	$2n$	n	0
(LogDisag)	$2\lceil \log_2 n \rceil + 3$	$2n$	$\lceil \log_2 n \rceil$	0
(Ag)	$n+5$	$n+1$	n	0
(LogAg)	$2\lceil \log_2 n \rceil + 3$	$n+1$	$\lceil \log_2 n \rceil$	0
(Inc)	$2n+1$	n	$n-1$	0
(MC)	$2n+3$	n	n	0
(BinZigZag)	$2\lceil \log_2(n-1) \rceil + 3$	$n+1$	$\lceil \log_2(n-1) \rceil$	0
(IntZigZag)	$2\lceil \log_2(n-1) \rceil + 3$	$n+1$	0	$\lceil \log_2(n-1) \rceil$

name, or an index. These indices are then used as a reference, for instance to determine where a nonlinearity is used.

A nonlinearity is stored as an expression tree that includes tags for functions like power, products, and sums. An example for the nested expression $-\frac{10^6 \cdot i_1 \cdot i_2}{i_3 \cdot i_4}$ is given in Figure 6a, the corresponding expression tree is given in Figure 6b. For variable i_1 with index 0, there is an additional parameter, its coefficient of 10^6. The nonlinear term is used in the constraint with index 0, what is stated in line 7. In line 2, one can see how properties of the variables are stored: In the provided example, variable i_1 is an integer variable with $12 \leq i_1 \leq 60$. The index of 0 is implicitly given by the order in which the variables are defined. In line 11, variable i_1 is then used in the nonlinear expression. It would have also been possible to store parts of this nonlinearity as quadratic equations: Therin, only two variable indices are stored together with a coefficient. More information about the OSIL format is provided by Fourer et al. 2010.

```
<variables numberOfVariables="6">
 <var name="i1" type="I" lb="12" ub="60"/>
  [...]
</variables>
 [...]
<nonlinearExpressions numberOfNonlinearExpressions="1">
  <nl idx="0">
    <negate>
      <divide>
        <product>
          <variable idx="0" coef="1e6"/>
          <variable idx="1"/>
        </product>
        <product>
          <variable idx="2"/>
          <variable idx="3"/>
        </product>
      </divide>
    </negate>
  </nl>
</nonlinearExpressions>
```

(6a) Parts of the OSIL code from instance `gear4.osil`.



(6b) Expression tree for the nonlinear term

FIGURE 6. Representations for the expression $-\frac{10^6 \cdot i_1 \cdot i_2}{i_3 \cdot i_4}$. This expression is a part of an instance from the MINLPLib.

Our benchmark set consists of 306 instances. We use a subset of the problems provided in the MINLPLIB (Bussieck et al. 2003), depending on different selection criteria. On the one hand, all variables need to have a lower and upper bound, as, otherwise, we are not able to establish a piecewise linear approximation or relaxation. On the other hand, as mentioned in Section 2, a large subset of the instances from the MINLPLIB can be represented by a recursive combination of elementary operators from (1). Based on this, we only consider the following nonlinear functions:

$$\begin{aligned} &f(x_1,\dots,x_n) = \prod_{i=1}^{n} x_i,\ f(x) = e^x,\ f(x) = \ln x, \\ &f(x) = \log_{10} x,\ f(x) = x^2,\ f(x) = \sqrt{x},\ f(x) = \sin x, \\ &f(x) = \cos x,\ f(x) = \tanh x,\ f(x_1,x_2) = \frac{x_1}{x_2},\ f(x) = x^{-1}. \end{aligned} \tag{30}$$

Furthermore, we include only problems where the optimal solution is known. In our numerical study in Section 5, we provide additional statistics about our benchmark instances.

4.2. **Reformulation.** Each input problem is now reformulated using a self-defined data format, first to a MINLP containing only one-dimensional nonlinearities and nonlinearities of the form $z = x_1x_2$. Subsequently, the univariate transformation of bivariate products from Section 2 is performed. The resulting MINLP then serves as the basis for the various MIP relaxations.

Our data format is called `OSILData` and contains data structures for variables, objective(s), constraints, linear, quadratic, and nonlinear expressions. Each object of `OSILData` can later be solved in the same way, independently of the type of nonlinearities, what makes the results easily comparable. All nonlinear expressions are stored in a tree structure that can be easily modified. This structure is based on the expression trees of the original OSIL format. At the start of each solving process, an object of `OSILData` containing all given information is created from the initial MINLP.

4.2.1. *Creating one-dimensional nonlinearities.* The first reformulation step is to remove all more-dimensional nonlinearities. To this end, we investigate all nonlinearities recursively and reformulate each type one by one. We initially use two-dimensional multiplications, which we reformulate at the end. The first more-dimensional nonlinearities that we want to remove are products with more than two multiplicands, i.e., expressions of the form $\texttt{product}(x_1, x_2, \ldots, x_{n-1}, x_n)$. This is reformulated such that each product has two multiplicands, resulting in the form $\texttt{product}(x_1,\texttt{product}(x_2,\texttt{product}(\ldots,\texttt{product}(x_{n-1}, x_n))))$. Next, each division of the form $z = \frac{x_1}{x_2}$ is reformulated by $z = x_1 \cdot \frac{1}{x_2}$, which is again a product of one-dimensional nonlinearities.

After dealing with these special cases, we need to consider nested nonlinear equations with multiple variables. The OSIL expression tree format is useful in this regard: We go through the tree recursively, replacing each nonlinearity $\text{nl}_i(x)$ with a newly introduced variable z_i. In addition, we insert a new constraint $z_i = \text{nl}_i(x)$, which creates a new expression tree. As we begin to replace the leafs, each newly created tree has a depth of at most one.

4.2.2. *Reformulation from one-dimensional MINLPs to MIP relaxations.* We now relax the MINLP based on the one-dimensional formulation using the MIP representations described in Section 3. Our implementation allows one to choose between relaxations and approximations, but for this study we only consider piecewise linear relaxations. The following approach is essentially the same for each PWL model. Since we reformulated all nonlinearities to one-dimensional nonlinear functions, each expression has the form $z = f(x)$ with a bounded variable x, i.e., $x^- \leq x \leq x^+$. Therefore, we can create a piecewise linear approximation function $\bar{f} : [x^-, x^+] \to \mathbb{R}$ for each nonlinear expression and extend it to a relaxation afterwards. We use an adjustable value ϵ that bounds the maximum error in each segment, i.e., we enforce

$$\left|\bar{f}(x) - f(x)\right| \leq \epsilon \tag{31}$$

for all x with $x^- \leq x \leq x^+$. The size of ϵ controls the number of breakpoints and, thus, segments we use. A smaller error bound leads to an increasing number of breakpoints and vice versa. The set of breakpoints $x_0, x_1, \ldots, x_n$ is created as follows: Beginning with the lower bound x^-, we set $x_0 = x^-$ and calculate the next point x_1 that fulfills (31) for $x_0 \leq x \leq x_1$ when setting $\bar{f}(x) = m_0 x + t_0$ where m_0 and t_0 describe the linear function that connects $(x_0, f(x_0))$ and $(x_1, f(x_1))$. This procedure uses binary search and it is repeated until we reach a breakpoint x_n with $x_n \geq x^+$. The value of this point is then set to x^+. Finally, using all linear approximations, we obtain the PWL approximation

$$\bar{f}(x) = m_i x + t_i, \tag{32}$$

where m_i and t_i are chosen depending on x, i.e., such that $\bar{x}_{i-1} \leq x \leq \bar{x}_i$. We enforce continuous piecewise linear approximations by interpolation, i.e., we have $m_i \bar{x}_{i-1} + t_i = f(\bar{x}_{i-1})$ and $m_i \bar{x}_i + t_i = f(\bar{x}_i)$ for $i = 1, \ldots, n$. To create relaxations, we just add another variable, bounded by $-\epsilon \leq e \leq \epsilon$ that controls the maximum error and add it to the piecewise linear function $\bar{f}(x)$, i.e., we replace each nonlinearity by $\bar{f}(x) + e$.

In our implementation, we can choose which representation from Section 3 to use for the relaxation. Then, the respective constraints replace the occurring nonlinearities in the input problem.

4.3. **Solving.** Finally, we have a MIP model for each different representation that can be solved using state-of-the-art MIP solution methods. To accomplish this, we first convert the MIPs from our own data structure to Pyomo models, cf. Bynum et al. 2021; Hart et al. 2011. We use Pyomo because it allows us to easily switch between different solvers. Gurobi cannot be used for general MINLPs, but we want to use it later to solve the MIPs and compare the results with the original solution. We use Gurobi Optimizer version 9.1.2 (Gurobi Optimization, LLC 2023) to solve the MIPs using a time limit of four hours. Each problem is optimized on the NHR@FAU clusters using Intel Xeon Gold 6326 CPUs with four cores, a total of 32 GB RAM, and a base frequency of 2.9 GHz.

We store different information in each run. On the one hand, we examine the optimization result, which tells us whether a problem was solved to optimality, reached its time limit, or ran into an error. Infeasibility can only occur when we use approximations instead of relaxations.

On the other hand, we are interested to understand more about how problems are solved: We store the primal and dual bounds on a regular basis and therefore can monitor the gap over time. Furthermore, we store the time it took until the first feasible solution was found. In the following section 5, we present the numerical results and discuss it afterwards in Section 6.

5. Numerical results

In this section, we first present the benchmark set from the MINLPLIB and then numerically analyze how the different MIP models for PWL relaxations from Section 3 perform in practice.

5.1. **Benchmark instances.** As mentioned previously, our benchmark set is a subset of the MINLPLIB that consists of 306 different instances. In Figure 7, one can see the numbers of constraints and variables for each instance. Figure 7a shows the entire benchmark set, whereas Figure 7b describes all instances with less than 1000 constraints and variables. In both plots, each blue square represents the total number of constraints and variables in one benchmark instance. We have around 169 variables and 225 constraints on average per instance. The median numbers are 42.5 and 30.5, respectively.

On the left hand side, we further plotted red circles that consider only the nonlinear constraints. Similarly, on the right hand side, the green circles show only the number of binary and integer variables. There are, on average, around 91 binary/integer variables, as well as 56 nonlinear constraints per instance. The median number of binary/integer variables is 4.5.

Table 2 contains statistics about the benchmark instances. To make a conclusive comparison, we consider the set of instances for which a model could be created for all error bounds. The number of segments per instance and nonlinearity, as well as the segment length per nonlinearity, are calculated for each error bound. The mean and median number and length of segments per nonlinearity are calculated for the entire set of nonlinearities instead of averaging over each instance separately.

A full list of the benchmark set and the exact numbers of variables and constraints per instance is provided in Appendix B.

5.2. **Comparison of the MIP models for PWL relaxations.** We now present the computational results where we test all PWL MIP models from Section 3 on the previously explained benchmark set. First, we evaluate how many problems can be solved to optimality, and how long the solving process takes before moving on to evaluate the solution quality over time.

For many results, we use the so-called shifted geometric mean (SGM). The SGM of n numbers $t_1, \dots, t_n$ is determined using the formula

$$\sqrt[n]{\prod_{i=1}^{n} (t_i + s)} - s. \tag{33}$$

Here, s represents an arbitrary shift applied to each term. This shift factor introduces a level of flexibility into the calculation, allowing us to adjust the significance of each term in the dataset. In our case, we want to decrease the impact of small runtimes. To improve the numerical stability of the computation, we employ an alternative formulation, namely

$$\sqrt[n]{\exp\left(\sum_{i=1}^{n} \ln (t_i + s)\right)} - s = \exp\left(\frac{\sum_{i=1}^{n} \ln (t_i + s)}{n}\right) - s. \tag{34}$$

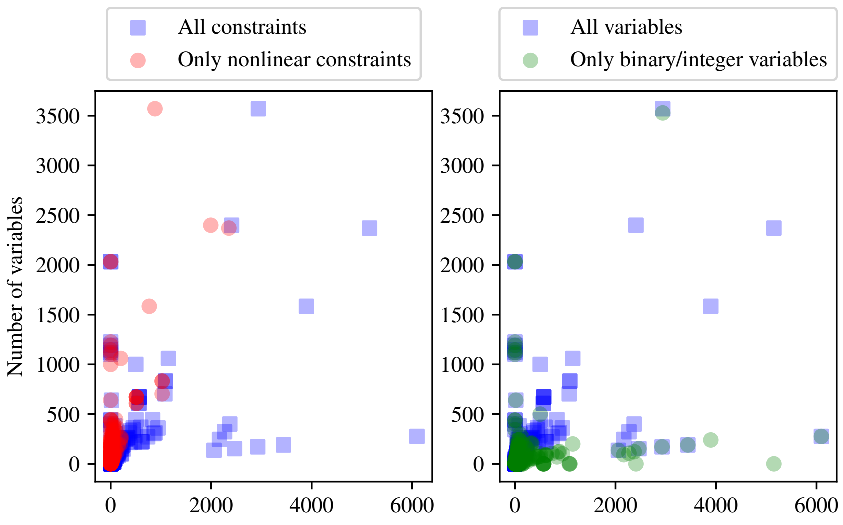
In each MIP relaxation, we fix a maximal error bound ϵ. The break points are created depending on this error bound. To allow the model to use the maximal error bound, we add the variables $\epsilon^+, \epsilon^- \in [0, \epsilon]$ every time, a non-linearity is replaced.

Table 2. Sizes of the MIP reformulations. The instances have between 2 and 12540 nonlinearities with a mean of 482.85 and a median of 48.

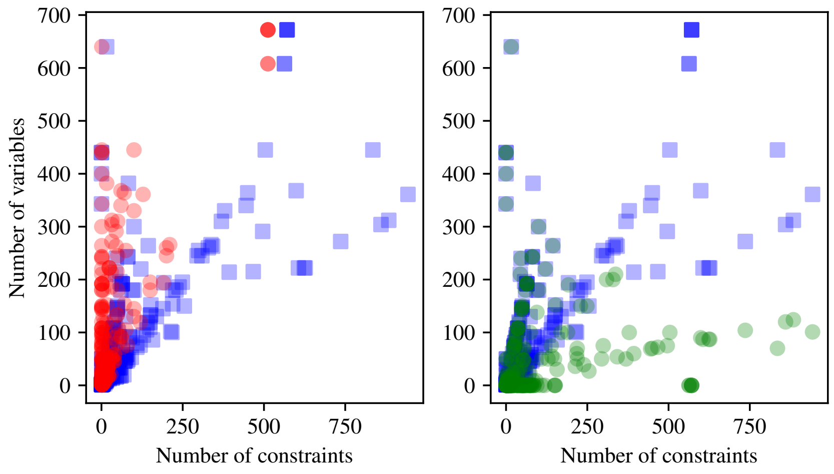
Error bound	$\epsilon = 10^2$			
	min	max	mean	median
Segments per instance	3	25079	1005.52	192
Segments per NL	1	173	1.08	1
Segment length per NL	0.0066	109669.00	7.97	1.00
Error bound	$\epsilon = 10^0$			
	min	max	mean	median
Segments per instance	3	25079	1534.06	669
Segments per NL	1	1729	2.18	1
Segment length per NL	0.0066	8436.08	1.94	1.00
Error bound	$\epsilon = 10^{-2}$			
	min	max	mean	median
Segments per instance	13	45343	9366.03	4589
Segments per NL	1	17281	18.40	5
Segment length per NL	0.0066	3456.00	0.34	0.20
Error bound	$\epsilon = 10^{-4}$			
	min	max	mean	median
Segments per instance	121	407098	88924.89	41414
Segments per NL	1	172801	183.17	50
Segment length per NL	0.0022	493.71	0.039	0.02
Error bound	$\epsilon = 10^{-6}$			
	min	max	mean	median
Segments per instance	1077	4029797	885031.81	413119.0
Segments per NL	4	1728349	1831.94	501
Segment length per NL	0.00022	51.58	0.0040	0.002

5.2.1. *Number of solved problems and runtimes.* First, we consider how many problems the various methods can solve. Figure 8 depicts the number of problems solved by each PWL model over time. We plot the number of optimally solved problems for each point in time between 0 and 14400 seconds (the time limit of 4 hours) and each method. The time is plotted on a logarithmic scale for better visibility. Each subfigure represents a different error bound (From now on, we will consider error bounds $\epsilon \in \{10^2, 10^0, 10^{-2}, 10^{-4}, 10^{-6}\}$ in all cases). All eight methods are plotted, and a legend is provided in Figure 8b. The bottom plots show more differences as the problems become larger and thus more difficult with increasing error bounds.

In Table 3, we compare the runtimes until an optimal solution is found. If an instance is not solved during the time limit, the time limit is used as



(7A) Full benchmark set

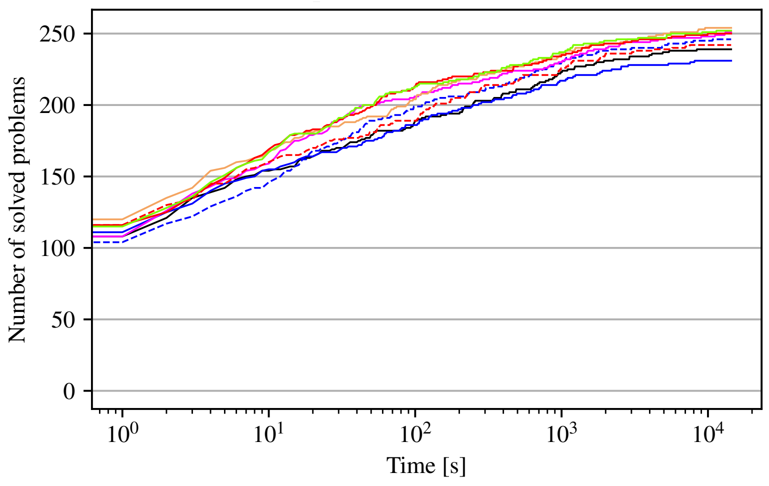


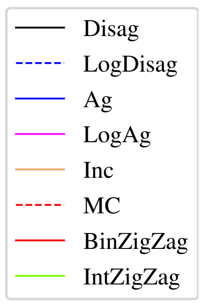
(7B) Instances with less than 1000 variables and constraints

FIGURE 7. Total amount of variables and constraints for our selected benchmark instances. The blue squares represent the total number of constraints and variables, whereas the red and green circles represent nonlinear constraint and binary/integer variable subsets, respectively.

a runtime. Additional to the mean and median values, we use the shifted geometric mean (SGM) with $s = 10$, as proposed in (33).

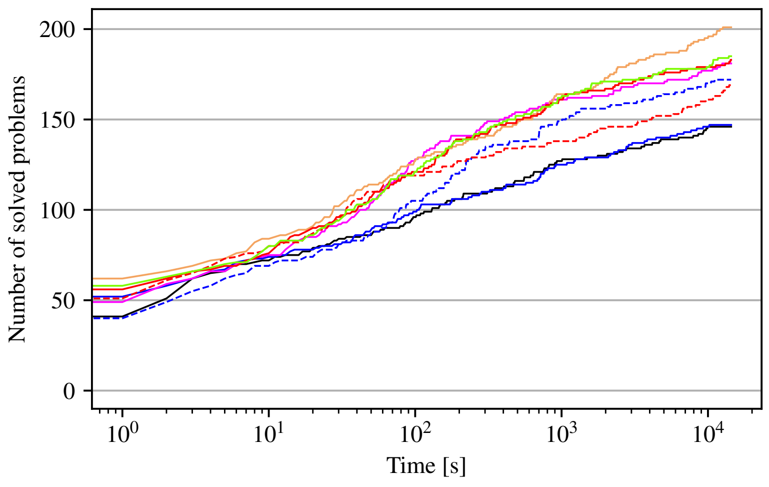
Table 4 distinguishes the instances according to whether they were solved within the time limit or not. As expected, the number of problems solved to optimality decreases as the error bounds become smaller. In some cases,





(8B) Legend

(8A) $\epsilon = 10^0$



(8C) $\epsilon = 10^{-2}$

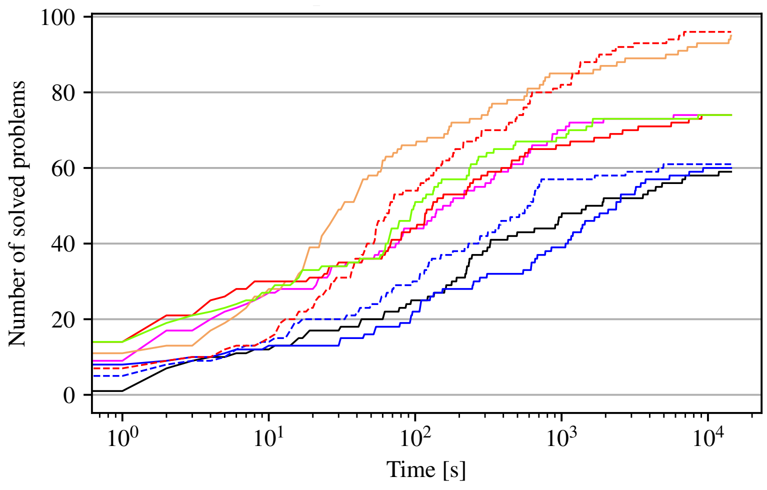
we do not have information about the solution process for every method because of errors that occur during the steps of our framework as described in Section 4: On the one hand, the model creation time may be too long, and the solution process may not begin at all; on the other hand, when models become too large, some out-of-memory errors occur. These cases are omitted from our statistics. Figure 9 displays how many instances of each MIP model progressed how far in the solution process. For every model, we plot a bar for each of the five error bounds. The different shades of blue represent the stages of the solution process, where the different checkpoints are the following: The whole process has started (STARTED), the MIP reformulation has been created (MIP CREATED), the MIP reformulation solution process has ended without errors (MIP SOLVED), and a primal or optimal solution of the relaxation has been found (PRIMAL/OPTIMAL SOLUTION FOUND), respectively.

Table 3. Mean, median, and shifted geometric mean of the runtimes, depending on the error bound ϵ.

Error bound	$\epsilon = 10^2$			$\epsilon = 10^0$		
Runtime [s]	mean	median	SGM	mean	median	SGM
(Disag)	781.88	0.05	12.74	1933.86	3.33	56.06
(LogDisag)	593.31	0.07	11.34	1596.81	5.90	46.98
(Ag)	732.12	0.06	11.59	2327.03	3.38	62.05
(LogAg)	577.94	0.05	10.05	1409.95	2.81	37.23
(Inc)	478.16	0.04	9.46	1197.74	2.07	34.34
(MC)	751.85	0.06	11.73	1760.72	3.11	47.99
(BinZigZag)	594.41	0.05	10.01	1335.04	2.96	33.49
(IntZigZag)	607.02	0.04	10.19	1251.36	2.93	32.52
Error bound	$\epsilon = 10^{-2}$			$\epsilon = 10^{-4}$		
Runtime [s]	mean	median	SGM	mean	median	SGM
(Disag)	6523.66	924.23	620.00	7703.17	14400.00	1004.48
(LogDisag)	5091.59	222.42	402.49	6258.89	1238.93	761.39
(Ag)	6444.64	1209.32	604.09	7533.72	9718.85	1170.59
(LogAg)	4623.80	93.69	281.33	4758.82	556.63	462.13
(Inc)	3778.40	98.03	235.87	4640.37	383.14	331.88
(MC)	5611.26	193.02	393.24	5249.12	104.08	361.25
(BinZigZag)	4494.56	136.44	277.48	5038.51	824.10	482.58
(IntZigZag)	4408.71	118.55	270.26	4950.96	427.57	416.71
Error bound	$\epsilon = 10^{-6}$					
Runtime [s]	mean	median	SGM			
(Disag)	7914.52	14400.00	1672.63			
(LogDisag)	7403.85	9675.72	1227.49			
(Ag)	7866.40	14400.00	1941.72			
(LogAg)	5839.51	500.07	569.06			
(Inc)	3783.14	59.33	227.57			
(MC)	3467.61	152.07	306.00			
(BinZigZag)	5995.32	453.34	574.04			
(IntZigZag)	5842.74	250.83	507.88			

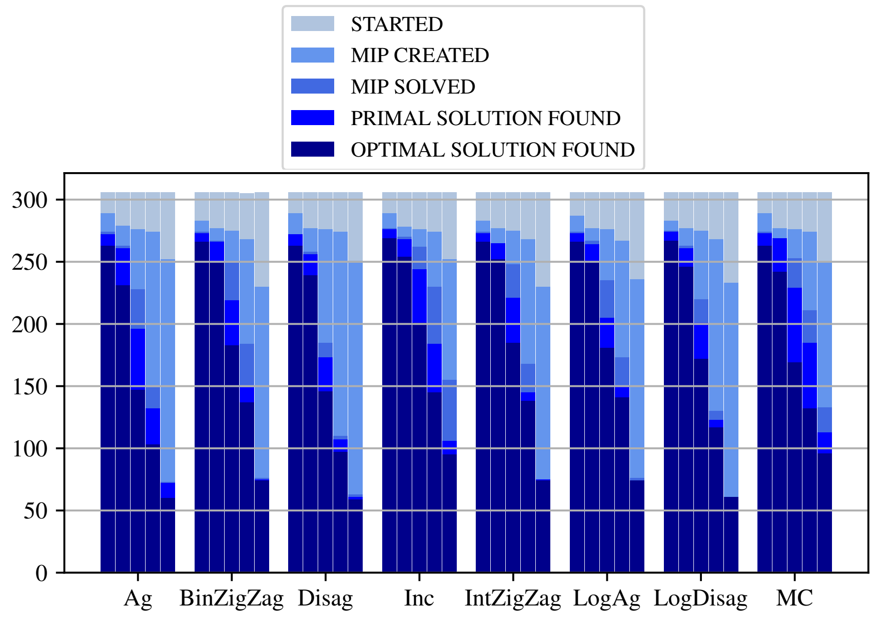
Table 4. Solver results for the full benchmark set.

Error bound	$\epsilon = 10^2$		$\epsilon = 10^0$		$\epsilon = 10^{-2}$		$\epsilon = 10^{-4}$		$\epsilon = 10^{-6}$	
Solver result	opt.	tl.	opt.	tl.	opt.	tl.	opt.	tl.	opt.	tl.
(Disag)	263	13	239	32	146	107	97	101	59	63
(LogDisag)	267	9	246	25	172	81	117	81	61	61
(Ag)	263	13	231	40	147	106	103	95	60	62
(LogAg)	266	10	251	20	181	72	141	57	74	48
(Inc)	269	7	254	17	201	52	145	53	95	27
(MC)	263	13	242	29	169	84	132	66	96	26
(BinZigZag)	266	10	250	21	183	70	137	61	74	48
(IntZigZag)	266	10	252	19	185	68	138	60	74	48



(8D) $\epsilon = 10^{-6}$

FIGURE 8. Number of solved problems over time. The remaining error bounds are given in Figure 11.



(9) Solving progress of the different MIP methods. For each method, the bars represent the error bounds of $10^2, 10^0, 10^{-2}, 10^{-4}$, and 10^{-6}, from left to right.

More benchmarks for smaller subsets, i.e., on the one hand, all problems for which a feasible solution was found, and on the other hand, all problems that are solved to optimality by all PWL models are given in Appendix C. Additionally, we present performance profiles in Appendix C.4.

Table 5. Solution qualities of MIP relaxations, depending on the error bound ϵ, calculated by (35).

Error bound	$\epsilon = 10^{2}$	$\epsilon = 10^{0}$	$\epsilon = 10^{-2}$	$\epsilon = 10^{-4}$	$\epsilon = 10^{-6}$
Solved instances	262	228	131	87	48
Mean gap	1517.84%	35.14%	7.85%	1.29%	2.25%
Median gap	9.60%	1.35%	0.01%	0.00%	0.00%
SGM gap	5.13%	1.83%	0.31%	0.03%	0.03%

5.2.2. *Solution qualities.* We can further evaluate the gap between the optimal solution of the MINLP and the MIP relaxations' solution for the instances that all PWL models solved to global optimality. Table 5 presents the corresponding results. Therein, additionally to the number of solved problems, we provide the mean, median, and shifted geometric mean for the gaps between the optimal MINLP and MIP solutions x^* and x^{relax}, given by

$$\frac{\left|c^\top x^* - c^\top x^{\text{relax}}\right|}{|c^\top x^*| + 10^{-10}}. \tag{35}$$

Adding 10^{-10} to the denominator prevents problems with instances that have an objective value of 0. As one can see, the gap is already very small for $\epsilon = 10^{-4}$, which means that using MIP relaxations, one can find fairly good dual bounds.

Further, in Figure 10, we plot how the MIP solution improves over time. Therefore, we take the shifted geometric mean of all primal and dual gaps, i.e., the gaps between the primal/dual bounds and the optimal MIP solution. The primal gap is calculated using

$$p := \begin{cases} 1 & \text{if } \overline{z} = \infty \text{ or } \overline{z} \cdot z^* < 0, \\ 0 & \text{if } \overline{z} = z^*, \\ \frac{|\overline{z} - z^*|}{\max\{|\overline{z}|, |z^*|\}} & \text{else,} \end{cases} \tag{36}$$

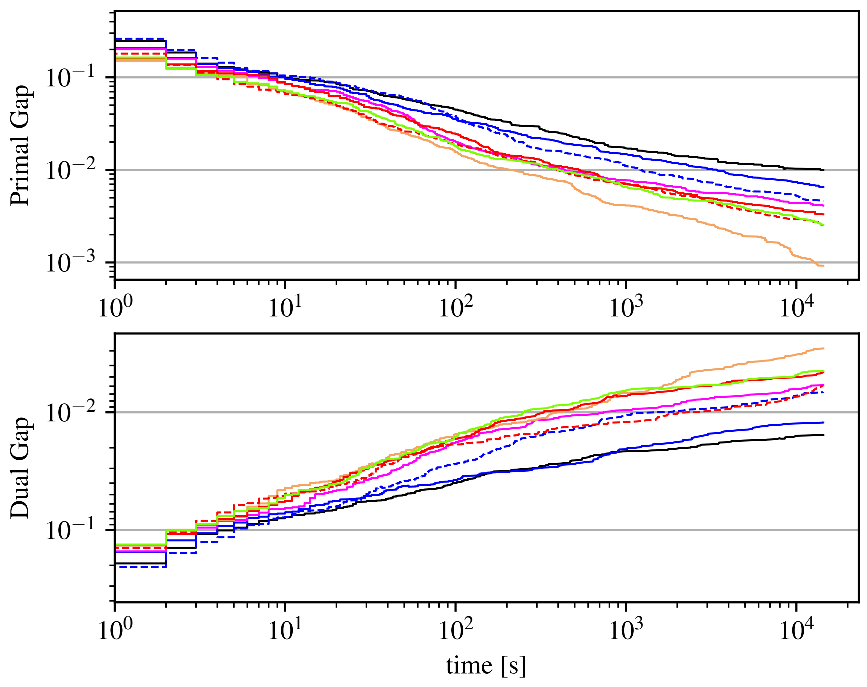
and, equivalently, the dual gap is calculated by

$$d := \begin{cases} 1 & \text{if } \underline{z} = -\infty \text{ or } \underline{z} \cdot z^* < 0, \\ 0 & \text{if } \underline{z} = z^*, \\ \frac{|\underline{z} - z^*|}{\max\{|\underline{z}|, |z^*|\}} & \text{else,} \end{cases} \tag{37}$$

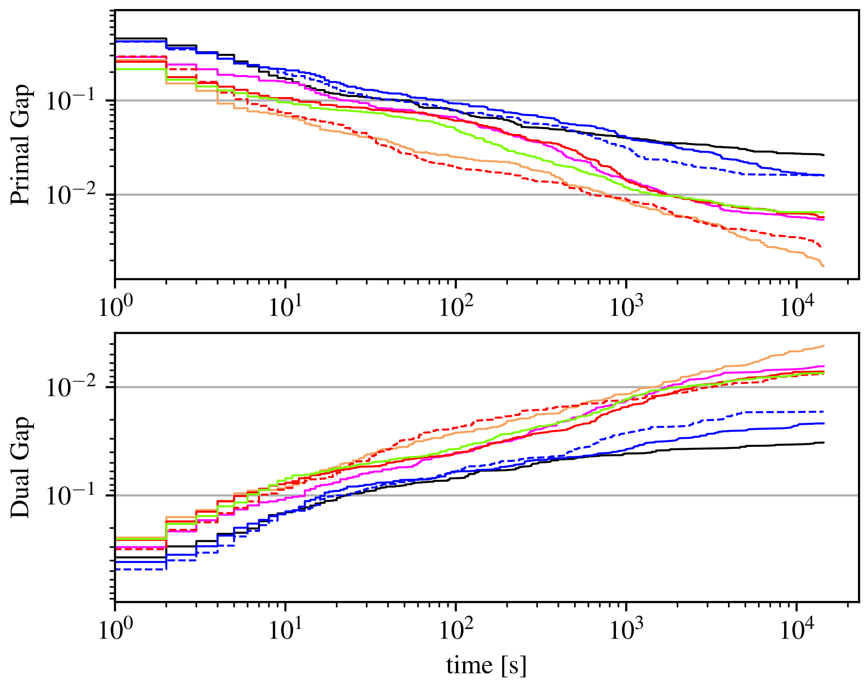
where $\overline{z}$ and $\underline{z}$ are the current primal and dual bounds and $z^* = c^\top x^{\text{relax}}$ is the optimal solution of the MIP relaxation. The upper graphs in each subplot show how the primal gap shrinks, while the lower graphs show how the dual gap shrinks.

6. Discussion

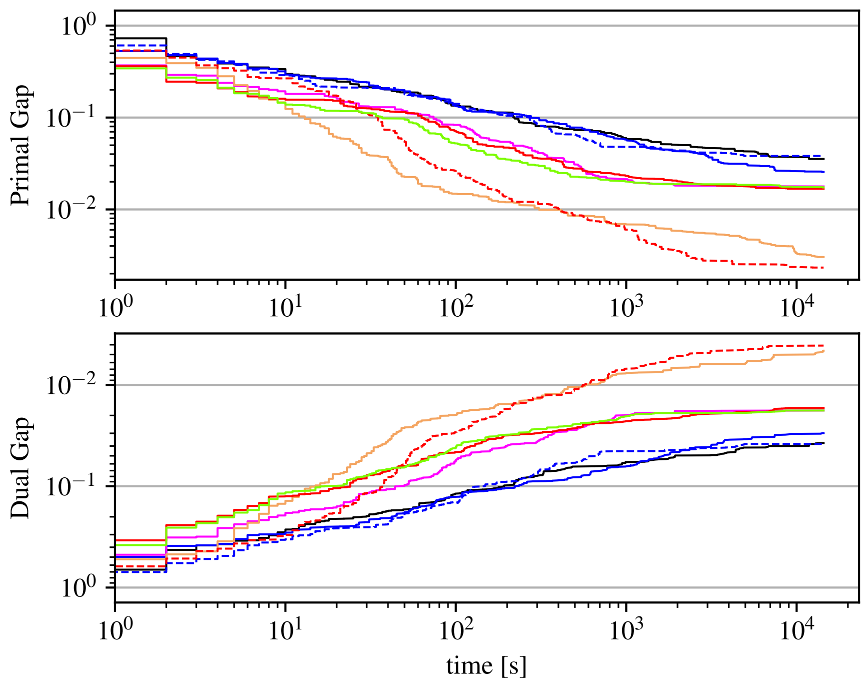
We will now investigate the presented results in order to draw some conclusions from our study. For larger error bounds (Figure 11a), the number of solved problems is quite similar, while already for $\epsilon = 1$ (Figure 8a), we can see some differences. For smaller error bounds, i.e., larger models, the



(10A) $\epsilon = 10^{-2}$



(10B) $\epsilon = 10^{-4}$



(10C) $\epsilon = 10^{-6}$

FIGURE 10. SGM of primal and dual gap to optimal MIP solution over time. The remaining error bounds are given in Figure 12

differences become more visible: For $\epsilon = 10^{-2}$ (Figure 8c), we see that (Inc), (BinZigZag), (IntZigZag), and (LogAg) show similar performances during the first 1000 seconds, but with more runtime, (Inc) solves significantly more problems. For $\epsilon = 10^{-6}$ (Figure 8d) it stands further out that (MC) also yields a good performance, catching up to (Inc) after around 1000 seconds and solving nearly the same number of problems after a time limit of 4 hours.

As a first result, we can say that the incremental method shows, overall, the best performance regarding the number of solved problems while, as we can see in all graphs of Figure 8 and in Table 4, the non-logarithmic convex combinations, (Ag) and (Disag), solve the fewest instances. In general, the logarithmic methods outperform their non-logarithmic counterparts, while the (IntZigZag) model is slightly superior to the other logarithmic models. The runtimes in Table 3 also strengthen our conclusions.

As mentioned in the previous section, we also considered two smaller subsets of instances: All problems for which a feasible solution was found by all methods and all instances that are solved to optimality by all models. For the first subset (Tables 8 and 9), we see similar results as before: (Inc) solves the most instances, followed by the logarithmic models. For error bounds $\epsilon \geq 10^{-4}$, (Inc) again shows superior runtimes. This changes with $\epsilon = 10^{-6}$, as now, (LogAg), (BinZigZag), and (IntZigZag) have shorter runtimes, while (IntZigZag) again performs best among the logarithmic methods. We can see the same results for the optimality subset, presented in Table 10, where (Inc) is also superior only up to $\epsilon = 10^{-4}$.

One reason for this seems to be that logarithmic methods are much less suitable for some of the more difficult instances (with very high accuracies), while for other problems they provide the best models overall. Therefore, depending on the specific instance, the logarithmic models can be a very valuable alternative to the incremental method for very small error bounds.

In Tables 11 to 13, smaller time limits are investigated, but the results coincide with our previous findings. As before, (Inc) solves the instances fastest, while the other methods catch up only for the easier subsets and very small error bounds.

Complementary to the runtimes, we also analyzed the solution process itself, i.e., how the optimality gaps decrease over time. However, before doing so, we investigated the extent to which our solution framework as described in Section 4 worked (Figure 9), where it stands out that (MC) is able to find primal solutions, i.e., solve feasibility problems quite fast. For larger error bounds (Figures 12a and 12b), the gaps evolve quite similar for all methods, while, for smaller error bounds (Figures 10a to 10c) the differences become greater. Only the non-logarithmic convex combinations again show worse results than the remaining methods. For all error bounds, the logarithmic versions of the aggregated convex combination, i.e., (LogAg), (BinZigZag), and (IntZigZag) show a similar behavior. This is not surprising, as these methods only differ in how the branching scheme is defined, see Section 3.2. Besides the fact, that (Inc) is again superior, we can further see that (MC) has also a good behavior for the very small bound $\epsilon = 10^{-6}$. Overall, our results lead to a recommendation to favor (Inc) as a general PWL method in practice over the other models that we considered in this paper.

Finally, to emphasize the relevance of PWL relaxations in the context of solving MINLPs, we discuss how good the MIP relaxation's solutions are in general. Since we know the optimal solution for each MINLP as we restricted ourselves to these instances, we can calculate the gap between the optimal MIP and MINLP solutions, as described in (35). Please note that the optimal solution of a MIP relaxation is in fact a valid dual bound for the corresponding MINLP. Thus, the results in Table 5, which shows the mean and median values of all gaps as well as the shifted geometric mean using a shift value of $s = 10^{-3}$, can be considered as relative optimality gaps for the MINLP problems. As we can see, the median gap is negligible already for $\epsilon = 10^{-2}$, which means that more than half of the MIP relaxation' solutions are similar to the corresponding optimal MINLP solution in terms of objective values. For $\epsilon \leq 10^{-4}$, also the mean and SGM values are fairly small. Obtaining such small gaps is a promising result, which suggests that tackling MINLPs by MIP relaxations can be a reasonable alternative to spatial branch-and-bound in practice. This becomes even more impressive if we consider that we have an expression tree structure with separate equations for each nonlinearity, which means that each nonlinearity can have errors up to ϵ that propagate further.

7. Conclusion

In this paper, we compared various MIP models for PWL relaxations of nonlinear functions that are known in the literature. Using over 300 instances of the MINLPLIB data set, we conducted a comprehensive computational study to determine a general performance of these models. Our results demonstrate the advantages of the incremental method as presented by Markowitz and Manne 1957 and are accompanied by a general recommendation of this method for practical applications.

Since our primary focus in this study was to evaluate different PWL models rather than assessing the overall performance of the solver, there are some points where our implementation could have been improved. For instance, a more sophisticated method for determining breakpoints or avoiding duplicates in variables and constraints would be desirable. It is important to note, however, that the inefficiencies in our framework remain consistent across all MIP models and do not affect the conclusions drawn from our research.

Future work may focus on different topics. First, this research only considers the maximum error for each segment, while overlooking factors such as the number of breakpoints and the specific characteristics of nonlinearities. Moreover, approaches that use PWL relaxations to solve MINLPs can benefit significantly from adaptivity by refining the PWL relaxations only locally in an iterative manner. Depending on the stages of the solution process, in this setting, PWL relaxations must thus be solved with both small and large numbers of segments. Consequently, the combination of different PWL models might be the best starting point for adaptive approaches, which is also supported by our results. One promising idea is to combine methods that are capable of finding primal solutions quickly, like the multiple choice method, with methods that are better suited for closing the optimality gap with high-accuracy PWL relaxations, like the incremental method or the integer Zig-Zag formulation by Huchette and Vielma 2022. Furthermore, when multiple kernels are available, the most suitable MIP model for a specific problem can be chosen during runtime.

Appendix A. Branching scheme for the logarithmic branching convex combination model

The following Table 6 gives a visualization of our branching scheme from Section 3.2.

Table 6. Example for the iterative creation of L_s, R_s for $n = 16$.

	$\bar{x}_0$	$\bar{x}_1$	$\bar{x}_2$	$\bar{x}_3$	$\bar{x}_4$	$\bar{x}_5$	$\bar{x}_6$	$\bar{x}_7$	$\bar{x}_8$	$\bar{x}_9$	$\bar{x}_{10}$	$\bar{x}_{11}$	$\bar{x}_{12}$	$\bar{x}_{13}$	$\bar{x}_{14}$	$\bar{x}_{15}$	$\bar{x}_{16}$
T_4	L_4	L_4	L_4	L_4	L_4	L_4	L_4	L_4	-	R_4	R_4	R_4	R_4	R_4	R_4	R_4	R_4
T_3	L_3	L_3	L_3	L_3	-	R_3	R_3	R_3	R_3	R_3	R_3	R_3	-	L_3	L_3	L_3	L_3
T_2	L_2	L_2	-	R_2	R_2	R_2	-	L_2	L_2	L_2	-	R_2	R_2	R_2	-	L_2	L_2
T_1	L_1	-	R_1	-	L_1	-	R_1	-	L_1	-	R_1	-	L_1	-	R_1	-	L_1

Appendix B. Benchmark set

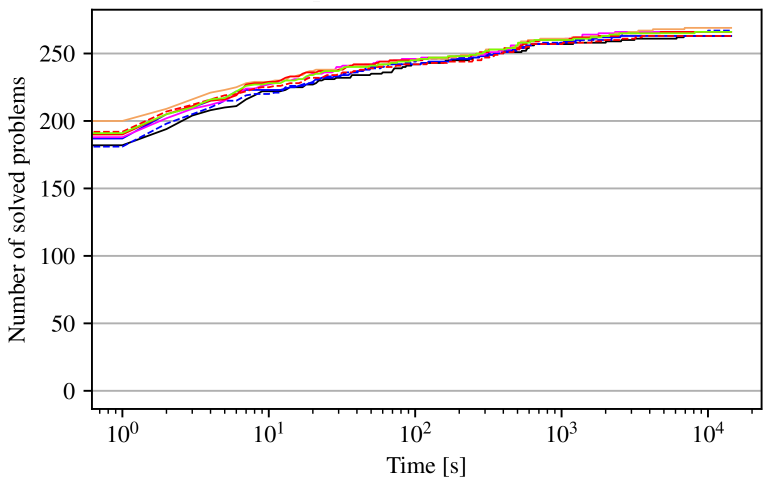
In the following, our benchmark set from the MINLPLIB is presented.

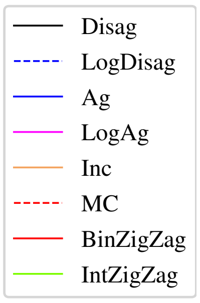
		Number of		Number of variables		
Instance	objective	constraints	nonlinear equations	continuous	binary	integer
alkyl	-1.765	7	7	14	0	0
arki0005	372.6047	5152	2353	2370	0	0
batch	285506.5082	73	2	22	24	0
batch0812	2687026.784	217	2	40	60	0
batchdes	167427.6571	19	2	10	9	0
blend029	13.3594	213	12	66	36	0
blend146	45.2966	624	24	135	87	0
blend480	9.2266	884	32	188	124	0
blend531	20.039	736	32	168	104	0
blend718	7.3936	606	24	135	87	0
blend721	13.5268	627	24	135	87	0
blend852	53.9627	860	32	184	120	0
cardqp_inlp	3760.7151	1	1	0	50	0
cardqp_iqp	3760.7151	1	1	0	50	0
celar6-sub0	159.0	16	1	0	640	0
chimera_k64ising-01	24.3	0	1	0	1192	0
chimera_k64ising-02	24.3	0	1	0	1225	0
chimera_k64maxcut-01	23.6	0	1	0	1101	0
chimera_k64maxcut-02	23.6	0	1	0	1145	0
chimera_lga-01	143.5	0	1	0	1120	0
chimera_mgw-c8-439-onc8-001	405.0	0	1	0	440	0
chimera_mgw-c8-439-onc8-002	397.0	0	1	0	440	0
chimera_mis-01	389.4	0	1	0	2032	0
chimera_mis-02	398.6	0	1	0	2032	0
chp_shorttermplan2a	245800.4121	3896	768	1344	240	0
crossdock_15x7	14409.0	44	1	0	210	0
crossdock_15x8	15595.0	46	1	0	240	0
cvxnonsep_normcon20	-21.7491	1	1	10	0	10
cvxnonsep_normcon30	-34.244	1	1	15	0	15
cvxnonsep_normcon40	-32.6297	1	1	20	0	20
du-opt	3.5563	9	1	7	0	13
ex1222	1.0765	3	2	2	1	0
ex1223	4.5796	13	5	7	4	0
ex1223a	4.5796	9	5	3	4	0
ex1223b	4.5796	9	5	3	4	0
ex1224	-0.9435	7	4	3	8	0
ex1263	19.6	55	4	20	72	0
ex1263a	19.6	35	4	0	4	20
ex1264	8.6	55	4	20	68	0
ex1264a	8.6	35	4	0	4	20
ex1265	10.3	74	5	30	100	0
ex1265a	10.3	44	5	0	5	30
ex1266	16.3	95	6	42	138	0
ex1266a	16.3	53	6	0	6	42
ex2_1_1	-17.0	1	1	5	0	0
ex2_1_5	-268.0146	11	1	10	0	0
ex2_1_6	-39.0	5	1	10	0	0
ex2_1_8	15639.0	10	1	24	0	0
ex3_1_1	7049.248	6	3	8	0	0
ex3_1_2	-30665.5387	6	7	5	0	0
ex5_2_2_case1	-400.0	6	3	9	0	0
ex5_2_2_case2	-600.0	6	3	9	0	0
ex5_2_2_case3	-750.0	6	3	9	0	0
ex5_2_4	-450.0	6	4	7	0	0
ex5_3_2	1.8642	16	9	22	0	0
ex5_4_2	7512.2301	6	3	8	0	0
ex6_2_14	-0.6954	2	1	4	0	0
ex7_2_1	1227.2261	14	13	7	0	0
ex8_1_1	-2.0218	0	1	2	0	0
ex8_4_1	0.6186	10	11	22	0	0
fac3	31982309.85	33	1	54	12	0
flay02m	37.9473	11	2	10	4	0
flay03m	48.9898	24	3	14	12	0
flay04m	54.4059	42	4	18	24	0
flay05m	64.4981	65	5	22	40	0
gabriel01	45.2444	467	48	143	72	0
gabriel04	9.2266	943	128	260	101	0
gbd	2.2	4	1	1	3	0
gear	0.0	0	1	0	0	4
gear2	0.0	4	1	4	24	0
gear3	0.0	4	1	4	0	4
genpooling_lee1	-4640.0824	82	20	40	9	0
genpooling_lee2	-3849.2654	92	30	44	9	0
genpooling_meyer04	1086187.137	141	15	63	55	0

graphpart_2g-0044-1601	-954077.0	16	1	0	48	0
graphpart_2g-0055-0062	-1484348.0	25	1	0	75	0
graphpart_2g-0066-0066	-2865560.0	36	1	0	108	0
graphpart_2g-0077-0077	-3282435.0	49	1	0	147	0
graphpart_2g-0088-0088	-5935341.0	64	1	0	192	0
graphpart_2g-0099-9211	-5070020.0	81	1	0	243	0
graphpart_2g-1010-0824	-7024864.0	100	1	0	300	0
graphpart_2pm-0044-0044	-13.0	16	1	0	48	0
graphpart_2pm-0055-0055	-20.0	25	1	0	75	0
graphpart_2pm-0066-0066	-29.0	36	1	0	108	0
graphpart_2pm-0077-0777	-40.0	49	1	0	147	0
graphpart_2pm-0088-0888	-55.0	64	1	0	192	0
graphpart_2pm-0099-0999	-64.0	81	1	0	243	0
graphpart_3g-0234-0234	-1952753.0	24	1	0	72	0
graphpart_3g-0244-0244	-2725383.0	32	1	0	96	0
graphpart_3g-0333-0333	-1882389.0	27	1	0	81	0
graphpart_3g-0334-0334	-3410696.0	36	1	0	108	0
graphpart_3g-0344-0344	-5301516.0	48	1	0	144	0
graphpart_3g-0444-0444	-7603756.0	64	1	0	192	0
graphpart_3pm-0234-0234	-20.0	24	1	0	72	0
graphpart_3pm-0244-0244	-28.0	32	1	0	96	0
graphpart_3pm-0333-0333	-26.0	27	1	0	81	0
graphpart_3pm-0334-0334	-36.0	36	1	0	108	0
graphpart_3pm-0344-0344	-48.0	48	1	0	144	0
graphpart_3pm-0444-0444	-65.0	64	1	0	192	0
graphpart_clique-20	147.0	20	1	0	60	0
graphpart_clique-30	495.0	30	1	0	90	0
graphpart_clique-40	1183.0	40	1	0	120	0
graphpart_clique-50	2312.0	50	1	0	150	0
graphpart_clique-60	3990.0	60	1	0	180	0
hmittelman	13.0	7	8	0	16	0
johnall	-224.7302	192	191	4	190	0
kall_circles_c6a	2.1117	54	22	18	0	0
kall_circles_c7a	2.6628	69	29	20	0	0
kall_circlespolygons_c1p11	0.1996	48	21	43	0	0
kall_circlesrectangles_c1r11	0.1996	52	23	49	0	0
kall_congruentcircles_c31	0.6438	16	4	10	0	0
kall_congruentcircles_c32	1.3759	16	4	10	0	0
kall_congruentcircles_c42	0.8584	24	7	12	0	0
kall_congruentcircles_c51	1.073	34	11	14	0	0
kall_congruentcircles_c52	1.5371	34	11	14	0	0
kall_congruentcircles_c62	1.2876	46	16	16	0	0
kall_congruentcircles_c63	1.2876	46	16	16	0	0
kall_congruentcircles_c71	1.5022	60	22	18	0	0
kall_congruentcircles_c72	1.9663	60	22	18	0	0
kall_diffcircles_5a	5.1162	24	11	14	0	0
kall_diffcircles_5b	5.1162	24	11	14	0	0
kall_diffcircles_6	7.7879	31	16	16	0	0
kall_diffcircles_7	7.1531	40	22	18	0	0
m6	82.2569	157	12	56	30	0
mathopt5_2	-1.0	0	1	1	0	0
mathopt5_3	-1.6164	0	1	1	0	0
mathopt5_5	-14.838	0	1	1	0	0
mathopt6	-3.3069	0	1	2	0	0
milinfract	2.6339	501	1	500	500	0
nvs01	12.4697	3	3	1	0	2
nvs02	5.9642	3	4	3	0	5
nvs03	16.0	2	2	0	0	2
nvs04	0.72	0	1	0	0	2
nvs07	4.0	2	2	0	0	3
nvs10	-310.8	2	3	0	0	2
nvs11	-431.0	3	4	0	0	3
nvs12	-481.2	4	5	0	0	4
nvs13	-585.2	5	6	0	0	5
nvs14	-40358.1548	3	4	3	0	5
nvs15	1.0	1	1	0	0	3
nvs17	-1100.4	7	8	0	0	7
nvs18	-778.4	6	7	0	0	6
nvs19	-1098.4	8	9	0	0	8
nvs20	230.9222	8	1	11	0	5
nvs23	-1125.2	9	10	0	0	9
nvs24	-1033.2	10	11	0	0	10
p_ball_10b_5p_2d_h	18.7186	219	50	130	50	0
p_ball_10b_5p_2d_m	18.7186	109	50	30	50	0
p_ball_10b_5p_3d_h	44.0042	294	50	195	50	0
p_ball_10b_5p_3d_m	44.0042	129	50	45	50	0
p_ball_10b_5p_4d_h	71.3719	369	50	260	50	0
p_ball_10b_5p_4d_m	71.3719	149	50	60	50	0

p_ball_10b_7p_3d_h	109.8032	450	70	294	70	0
p_ball_10b_7p_3d_m	109.8032	219	70	84	70	0
p_ball_15b_5p_2d_h	6.5999	299	75	180	75	0
p_ball_15b_5p_2d_m	6.5999	139	75	30	75	0
p_ball_20b_5p_2d_h	2.4372	379	100	230	100	0
p_ball_20b_5p_2d_m	2.4372	169	100	30	100	0
p_ball_20b_5p_3d_h	19.7365	504	100	345	100	0
p_ball_20b_5p_3d_m	19.7365	189	100	45	100	0
p_ball_30b_5p_2d_m	0.2916	229	150	30	150	0
p_ball_30b_5p_3d_m	8.2183	249	150	45	150	0
p_ball_30b_7p_2d_m	13.9338	337	210	56	210	0
p_ball_40b_5p_3d_m	9.7772	309	200	45	200	0
p_ball_40b_5p_4d_h	30.1327	1149	200	860	200	0
p_ball_40b_5p_4d_m	30.1327	329	200	60	200	0
pooling_adhya1pq	-549.8031	49	20	33	0	0
pooling_adhya1stp	-549.8031	71	40	46	0	0
pooling_adhya1tp	-549.8031	49	20	33	0	0
pooling_adhya2pq	-549.8031	57	20	33	0	0
pooling_adhya2stp	-549.8031	79	40	46	0	0
pooling_adhya2tp	-549.8031	57	20	33	0	0
pooling_adhya3pq	-561.0447	74	32	52	0	0
pooling_adhya3stp	-561.0447	109	64	72	0	0
pooling_adhya3tp	-561.0447	74	32	52	0	0
pooling_adhya4pq	-877.6457	77	40	58	0	0
pooling_adhya4stp	-877.6457	119	80	76	0	0
pooling_adhya4tp	-877.6457	77	40	58	0	0
pooling_bental4pq	-450.0	16	6	13	0	0
pooling_bental4stp	-450.0	23	12	18	0	0
pooling_bental4tp	-450.0	16	6	13	0	0
pooling_bental5pq	-3500.0	86	60	92	0	0
pooling_bental5stp	-3500.0	149	120	119	0	0
pooling_bental5tp	-3500.0	86	60	92	0	0
pooling_foulds2pq	-1100.0	34	16	36	0	0
pooling_foulds2stp	-1100.0	52	32	48	0	0
pooling_foulds2tp	-1100.0	34	16	36	0	0
pooling_foulds3pq	-8.0	571	512	672	0	0
pooling_foulds3stp	-8.0	1091	1024	832	0	0
pooling_foulds3tp	-8.0	571	512	672	0	0
pooling_foulds4pq	-8.0	571	512	672	0	0
pooling_foulds4stp	-8.0	1091	1024	832	0	0
pooling_foulds4tp	-8.0	571	512	672	0	0
pooling_foulds5pq	-8.0	563	512	608	0	0
pooling_foulds5stp	-8.0	1079	1024	704	0	0
pooling_foulds5tp	-8.0	563	512	608	0	0
pooling_haverly1pq	-400.0	13	4	10	0	0
pooling_haverly1stp	-400.0	18	8	14	0	0
pooling_haverly1tp	-400.0	13	4	10	0	0
pooling_haverly2pq	-600.0	13	4	10	0	0
pooling_haverly2stp	-600.0	18	8	14	0	0
pooling_haverly2tp	-600.0	13	4	10	0	0
pooling_haverly3pq	-750.0	13	4	10	0	0
pooling_haverly3stp	-750.0	18	8	14	0	0
pooling_haverly3tp	-750.0	13	4	10	0	0
pooling_rt2pq	-4391.8259	52	18	34	0	0
pooling_rt2stp	-4391.8259	72	36	46	0	0
pooling_rt2tp	-4391.8259	52	18	34	0	0
pooling_sppa9tp	-21933.994	2407	1992	2399	0	0
prob02	112235.0	8	5	0	0	6
prob03	10.0	1	1	0	0	2
prob06	1.1771	2	2	2	0	0
prob10	3.4455	2	1	1	0	1
process	-1161.3366	7	5	10	0	0
qp3	0.0008	52	1	100	0	0
qspp_0_10_0_1_10_1	621.0	100	1	0	180	0
qspp_0_11_0_1_10_1	813.0	121	1	0	220	0
qspp_0_12_0_1_10_1	959.0	144	1	0	264	0
rbrock	0.0	0	1	2	0	0
sep1	-510.081	31	6	27	2	0
sfacloc2_2_80	13.2795	2165	30	154	92	0
sfacloc2_2_90	18.5941	393	30	154	60	0
sfacloc2_2_95	19.5776	239	30	147	39	0
sfacloc2_3_80	11.0585	2268	45	216	107	0
sfacloc2_3_90	15.0945	496	45	216	75	0
sfacloc2_3_95	16.1511	342	45	209	54	0
sfacloc2_4_80	9.9531	2371	60	278	122	0
sfacloc2_4_90	13.4115	599	60	278	90	0
sfacloc2_4_95	14.2992	445	60	271	69	0
sonet17v4	1182604.5	2057	17	0	136	0

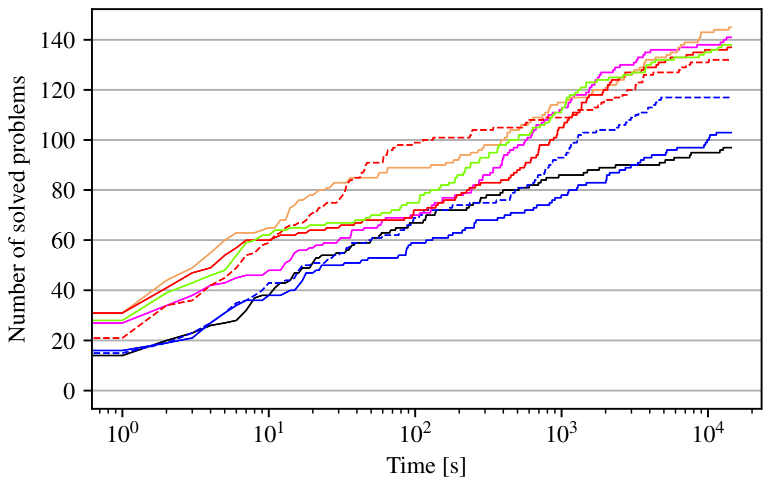
sonet18v6	3389110.0	2466	18	0	153	0
sonet19v5	2528144.0	2926	19	0	171	0
sonet20v6	3311060.0	3440	20	0	190	0
sonet24v2	3312579.0	6096	24	0	276	0
sonetgr17	-9594.0	152	17	0	152	0
st_bpaf1a	-45.3797	10	1	10	0	0
st_bpaf1b	-42.9626	10	1	10	0	0
st_bpv1	10.0	4	1	4	0	0
st_bpv2	-8.0	5	1	4	0	0
st_bsj3	-86768.55	1	1	6	0	0
st_bsj4	-70262.05	4	1	6	0	0
st_e01	-6.6667	1	1	2	0	0
st_e02	201.1593	3	3	3	0	0
st_e03	-1161.3366	7	5	10	0	0
st_e05	7049.2493	3	2	5	0	0
st_e07	-400.0	7	3	10	0	0
st_e08	0.7418	2	2	2	0	0
st_e09	-0.5	1	2	2	0	0
st_e13	2.0	2	1	1	1	0
st_e14	4.5796	13	5	7	4	0
st_e18	-2.8284	4	2	2	0	0
st_e22	-85.0	5	1	2	0	0
st_e23	-1.0833	2	1	2	0	0
st_e24	3.0	4	1	2	0	0
st_e26	-185.7792	4	1	2	0	0
st_e27	2.0	6	1	2	2	0
st_e29	-0.9435	7	4	3	8	0
st_e30	-1.5811	15	5	14	0	0
st_e33	-400.0	6	3	9	0	0
st_e34	0.0156	4	4	6	0	0
st_ht	-1.6	3	1	2	0	0
st_iqpbk1	-621.4878	7	1	8	0	0
st_iqpbk2	-1195.2256	7	1	8	0	0
st_jcbpaf2	-794.8559	13	1	10	0	0
st_miqp1	281.0	1	1	0	0	5
st_miqp2	2.0	3	1	0	0	4
st_miqp3	-6.0	1	1	0	0	2
st_miqp4	-4574.0	4	1	3	0	3
st_miqp5	-333.8889	13	1	5	0	2
st_test2	-9.25	2	1	0	0	6
st_test3	-7.0	10	1	0	0	13
st_test4	-7.0	5	1	0	0	6
st_test5	-110.0	11	1	0	0	10
st_test6	471.0	5	1	0	0	10
st_test8	-29605.0	20	1	0	0	24
st_testgr1	-12.8116	5	1	0	0	10
st_testgr3	-20.59	20	1	0	0	20
st_testph4	-80.5	10	1	0	0	3
supplychain	2260.2566	30	6	24	3	0
supplychainp1_020306	437551.6764	255	1	123	27	0
supplychainp1_030510	860440.9195	835	1	375	70	0
synthes3	68.0097	23	5	9	8	0
telecomsp_pacbell	310340.0	2940	882	42	3528	0
tln2	5.3	12	2	0	2	6
tln4	8.3	24	4	0	4	20
tln5	10.3	30	5	0	5	30
tln6	15.3	36	6	0	6	42
tln7	15.0	42	7	0	7	56
tloss	16.3	53	6	0	6	42
tltr	48.0667	54	3	0	12	36
toroidal2g20_5555	24838942.0	0	1	0	400	0
toroidal3g7_6666	33611981.0	0	1	0	343	0
trig	-3.7625	1	2	1	0	0
wastewater02m1	130.7025	14	3	19	0	0
wastewater02m2	130.7025	44	12	41	0	0
wastewater04m1	89.8361	21	6	23	0	0
wastewater04m2	89.8361	65	18	55	0	0
wastewater05m1	229.7008	40	12	46	0	0
wastewater05m2	229.7008	151	48	133	0	0
wastewater11m1	2127.1154	42	8	118	0	0
wastewater12m1	1201.0385	57	11	196	0	0
wastewater13m1	1564.958	83	16	382	0	0
wastewater15m1	2446.4286	40	12	46	0	0
wastewater15m2	2446.4286	151	48	133	0	0
waterund08	164.4898	95	40	90	0	0
waterund11	104.8861	64	28	64	0	0





(11b) Legend

(11a) $\epsilon = 10^2$



(11c) $\epsilon = 10^{-4}$

Figure 11. Number of solved problems over time for all considered error bounds.

Appendix C. Full results

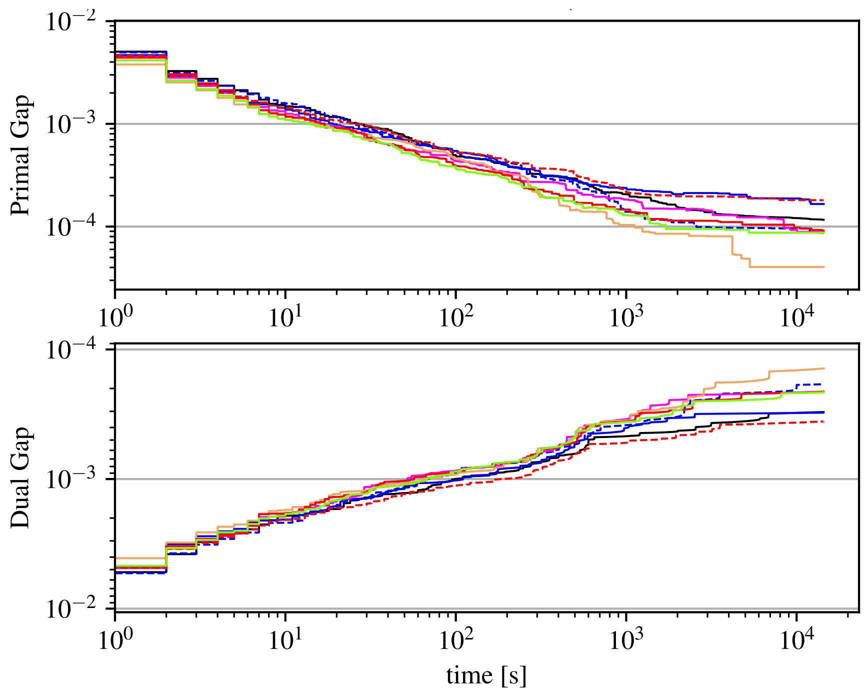
In this section, we present the remaining figures and tables that were omitted in Section 5 and further present performance profiles in Appendix C.4.

C.1. **Number of solved problems and runtimes.** This section contains the remaining plots from Section 5.2.1. In Figures 11a and 11c the remaining plots for Figure 8 are given.

Further, we consider instances where any method finds at least one feasible solution, as this creates a subset of easier instances for each error bound. The solving results for this subset can be seen in Table 8. Further, in Table 9, we again provide the runtimes as in Table 3. Finally, we want to consider

TABLE 8. Solver result for the subset of benchmark instances, where all reformulations found a feasible solution.

Error bound	$\epsilon = 10^2$		$\epsilon = 10^0$		$\epsilon = 10^{-2}$		$\epsilon = 10^{-4}$		$\epsilon = 10^{-6}$	
Solver result	opt.	tl.	opt.	tl.	opt.	tl.	opt.	tl.	opt.	tl.
(Disag)	263	8	238	13	143	21	93	7	51	2
(LogDisag)	265	6	242	9	145	19	98	2	53	0
(Ag)	263	8	229	22	134	30	92	8	49	4
(LogAg)	265	6	246	5	150	14	98	2	53	0
(Inc)	266	5	246	5	152	12	99	1	53	0
(MC)	263	8	240	11	151	13	94	6	53	0
(BinZigZag)	265	6	244	7	148	16	98	2	53	0
(IntZigZag)	265	6	245	6	148	16	98	2	53	0



(12A) $\epsilon = 10^2$

the subset of instances that were solved to optimality by all methods. The runtimes for this subset are given in Table 10.

C.2. **Solution qualities.** This section contains the remaining plots of Section 5.2.2. In Figures 12a and 12b the remaining plots for Figure 10 are given.

C.3. **Solving time for smaller time limits.** If we consider shorter time limits, we also provide statistics about the runtime. Tables 11 to 13 show the shifted geometric mean if the runtimes are limited to 100, 1000, and 10000 seconds, respectively. Every runtime that exceeds this time limit is then

TABLE 9. Mean, median, and shifted geometric mean of the runtimes, depending on the error bound ϵ, for the subset of benchmark instances, where all reformulations found a feasible solution.

Error bound	$\epsilon = 10^2$			$\epsilon = 10^0$		
Runtime [s]	mean	median	SGM	mean	median	SGM
(Disag)	530.63	0.05	10.19	982.15	2.30	33.23
(LogDisag)	431.57	0.07	9.40	793.57	3.62	28.47
(Ag)	479.94	0.05	9.15	1461.47	2.00	37.94
(LogAg)	375.95	0.05	8.15	651.16	1.99	22.11
(Inc)	351.85	0.03	7.64	460.96	1.40	19.48
(MC)	500.03	0.05	9.28	852.25	1.63	28.01
(BinZigZag)	392.72	0.04	8.11	607.16	2.22	19.61
(IntZigZag)	405.56	0.04	8.28	570.27	1.68	19.08
Error bound	$\epsilon = 10^{-2}$			$\epsilon = 10^{-4}$		
Runtime [s]	mean	median	SGM	mean	median	SGM
(Disag)	2410.16	27.36	108.68	1493.42	19.31	69.30
(LogDisag)	2001.03	32.05	80.60	626.46	17.98	56.60
(Ag)	2927.06	34.08	114.78	1803.96	27.65	107.08
(LogAg)	1648.93	15.32	53.89	425.42	12.72	35.13
(Inc)	1412.24	7.86	42.16	385.24	3.20	17.93
(MC)	1827.02	14.84	60.48	1019.52	6.45	27.62
(BinZigZag)	1733.46	12.30	54.37	581.44	4.21	34.84
(IntZigZag)	1737.36	11.55	54.29	571.71	5.20	31.59
Error bound	$\epsilon = 10^{-6}$					
Runtime [s]	mean	median	SGM			
(Disag)	1394.11	158.20	156.44			
(LogDisag)	180.86	63.86	65.88			
(Ag)	1949.63	133.73	232.07			
(LogAg)	60.56	9.63	22.83			
(Inc)	133.47	24.26	33.19			
(MC)	555.43	26.68	77.53			
(BinZigZag)	72.50	5.03	22.40			
(IntZigZag)	46.12	10.69	20.10			

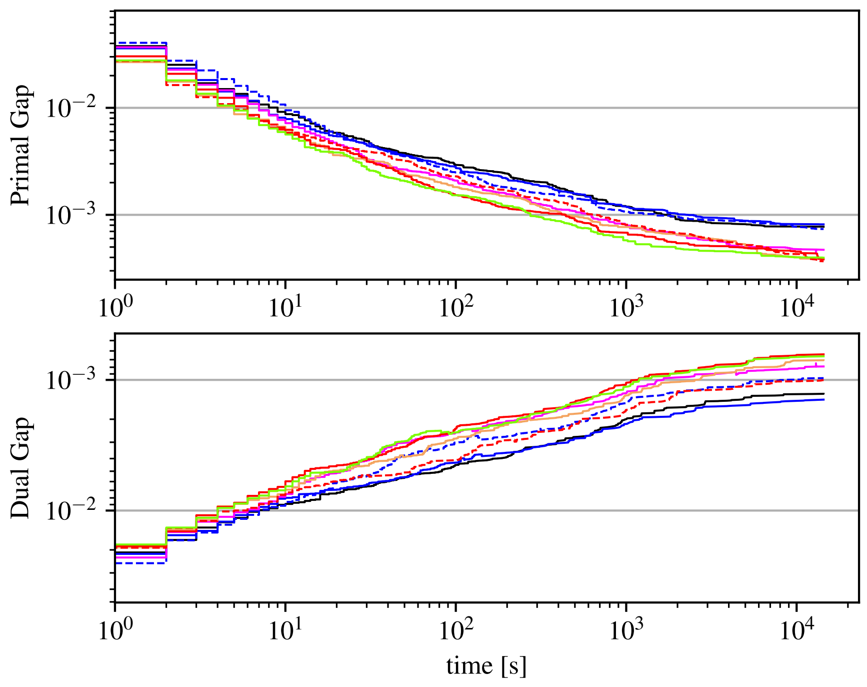
limited to the time limit. Table 11 considers the full instance that, while Table 12 and Table 13 are limited to our easier subsets.

C.4. **Performance profiles.** Additionally, we provide performance profile plots as proposed by Dolan and Moré 2002 to illustrate the scaling of the runtimes, see Figures 13 and 14. The intention here is to obtain a more sophisticated picture of how the various methods perform if we allow the runtime to lie within a given factor of the best overall runtime. The performance profiles work as follows: Let $t_{p,s}$ be the runtime needed by MIP relaxation $\bar{s}$ to solve instance p. With the performance ratio $r_{p,\bar{s}} := t_{p,\bar{s}} / \min_s t_{p,s}$, the

TABLE 10. Mean, median, and shifted geometric mean of the runtimes, depending on the error bound ϵ, for the subset of benchmark instances, where all reformulations found an optimal solution.

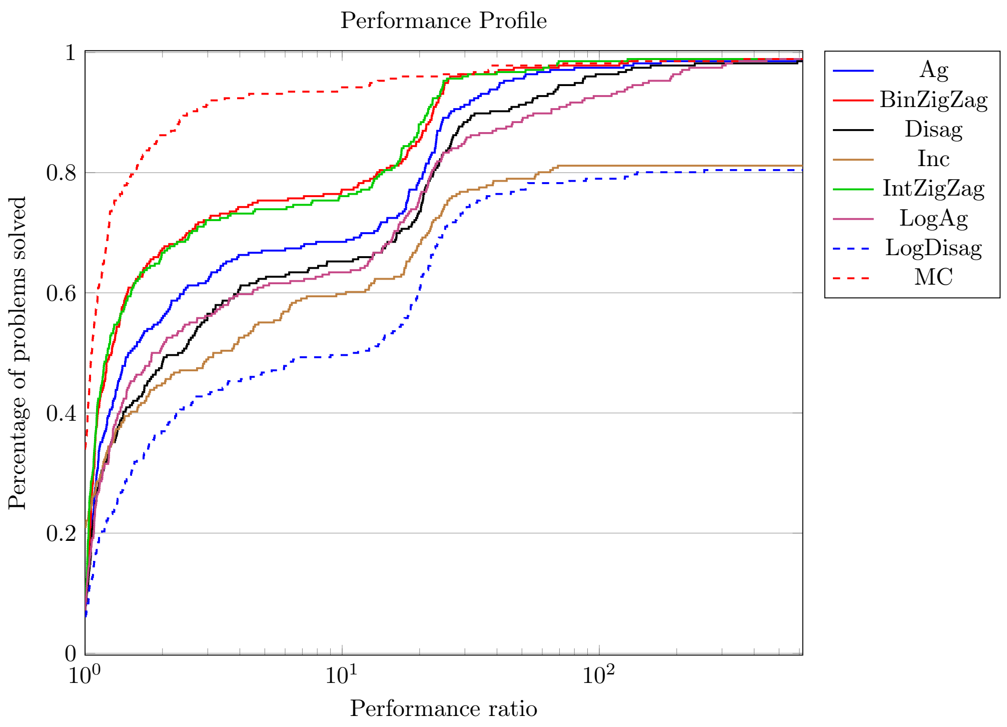
Error bound	$\epsilon = 10^2$			$\epsilon = 10^0$		
Runtime [s]	mean	median	SGM	mean	median	SGM
(Disag)	83.073	0.045	6.156	192.855	1.316	17.978
(LogDisag)	70.015	0.049	5.927	181.314	1.694	15.474
(Ag)	51.969	0.044	5.392	216.877	1.076	17.345
(LogAg)	52.550	0.042	5.095	151.047	1.087	11.760
(Inc)	50.911	0.029	4.661	143.833	0.865	11.596
(MC)	64.007	0.043	5.441	184.090	0.906	14.981
(BinZigZag)	53.929	0.040	4.925	149.091	0.955	10.933
(IntZigZag)	51.300	0.040	4.968	138.258	0.996	10.766
Error bound	$\epsilon = 10^{-2}$			$\epsilon = 10^{-4}$		
Runtime [s]	mean	median	SGM	mean	median	SGM
(Disag)	390.092	4.558	35.233	447.796	14.443	36.856
(LogDisag)	293.017	7.382	28.307	214.715	12.882	34.803
(Ag)	292.547	3.850	29.684	497.514	16.996	52.234
(LogAg)	145.863	3.804	18.455	104.550	5.002	22.205
(Inc)	61.531	1.816	12.808	40.322	1.877	7.845
(MC)	120.356	3.195	15.681	84.181	4.186	11.059
(BinZigZag)	150.772	2.825	18.779	283.115	2.309	22.721
(IntZigZag)	207.481	2.409	18.428	193.467	3.342	19.981
Error bound	$\epsilon = 10^{-6}$					
Runtime [s]	mean	median	SGM			
(Disag)	889.824	107.748	119.516			
(LogDisag)	176.487	61.299	62.140			
(Ag)	947.495	109.116	161.965			
(LogAg)	61.106	6.942	21.705			
(Inc)	124.841	18.438	27.190			
(MC)	439.842	23.224	60.710			
(BinZigZag)	73.554	3.753	21.333			
(IntZigZag)	44.949	6.897	18.321			

performance profile function value $P(\tau)$ is the percentage of problems solved by approach $\bar{s}$ such that the ratios $r_{p,\bar{s}}$ are within a factor $\tau \in \mathbb{R}$ of the best possible ratios. Figure 13 considers the time until a first feasible solution was found while Figure 14 considers the overall runtime until an instance was solved to optimality. All performance profiles are generated with the help of *perprof-py* (Siqueira et al. 2016).

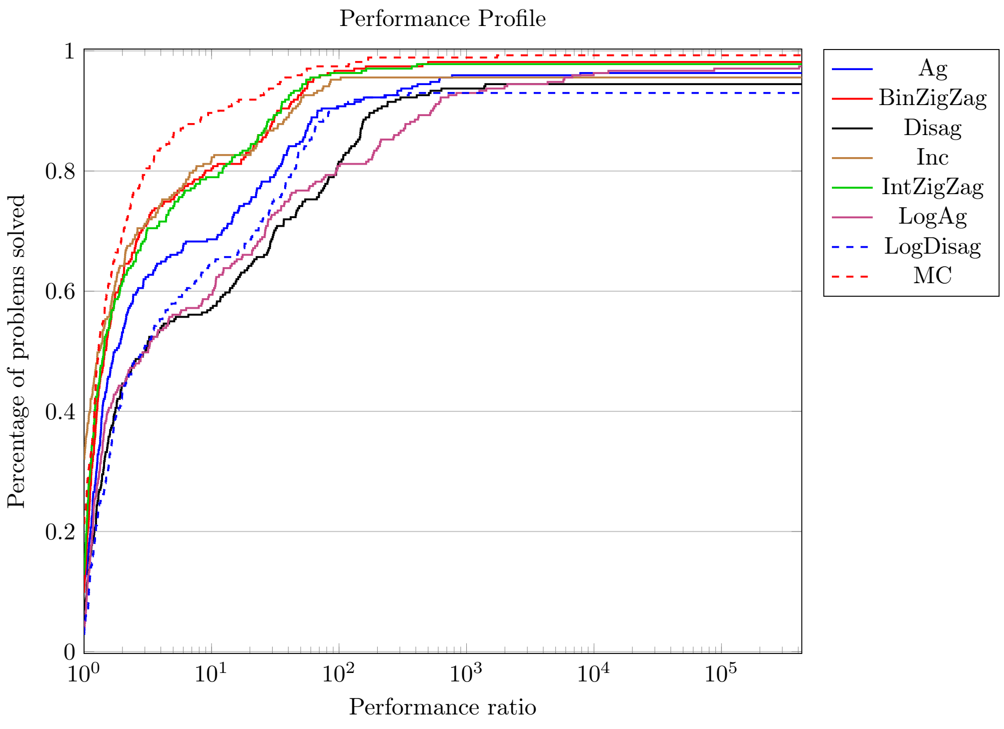


(12B) $\epsilon = 10^0$

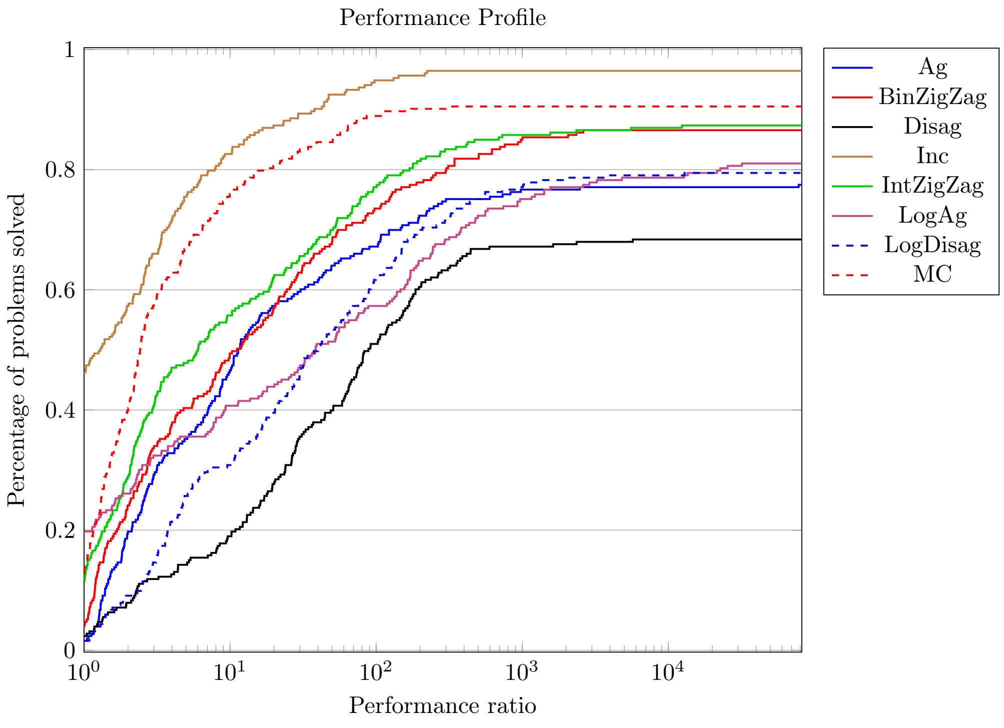
FIGURE 12. SGM of primal and dual gap to optimal MIP solution over time.



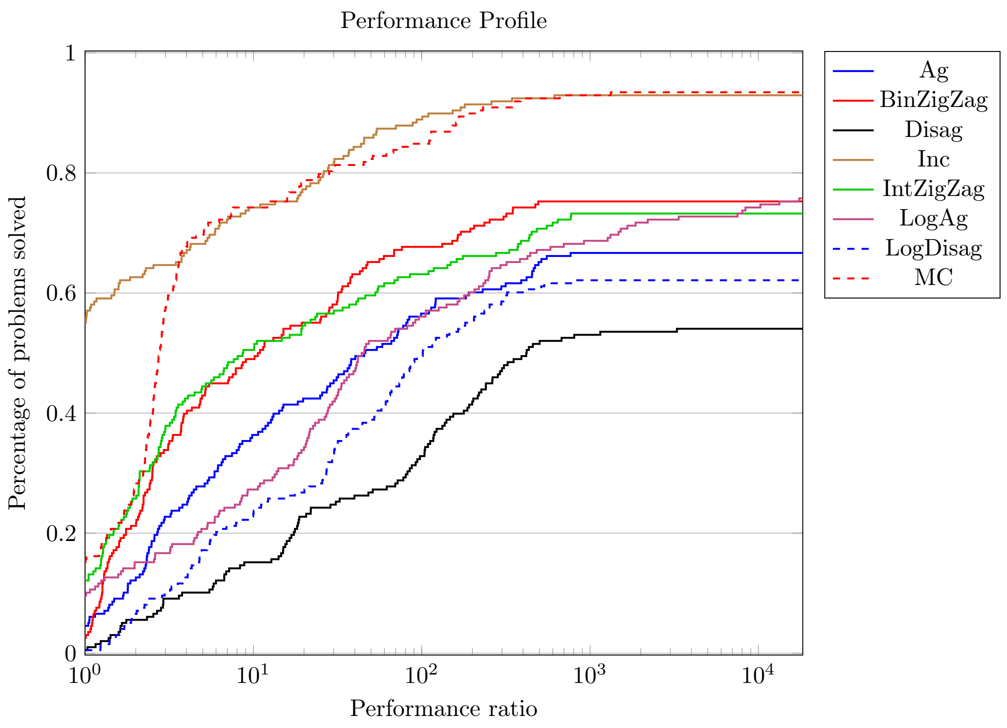
(13A) $\epsilon = 10^2$



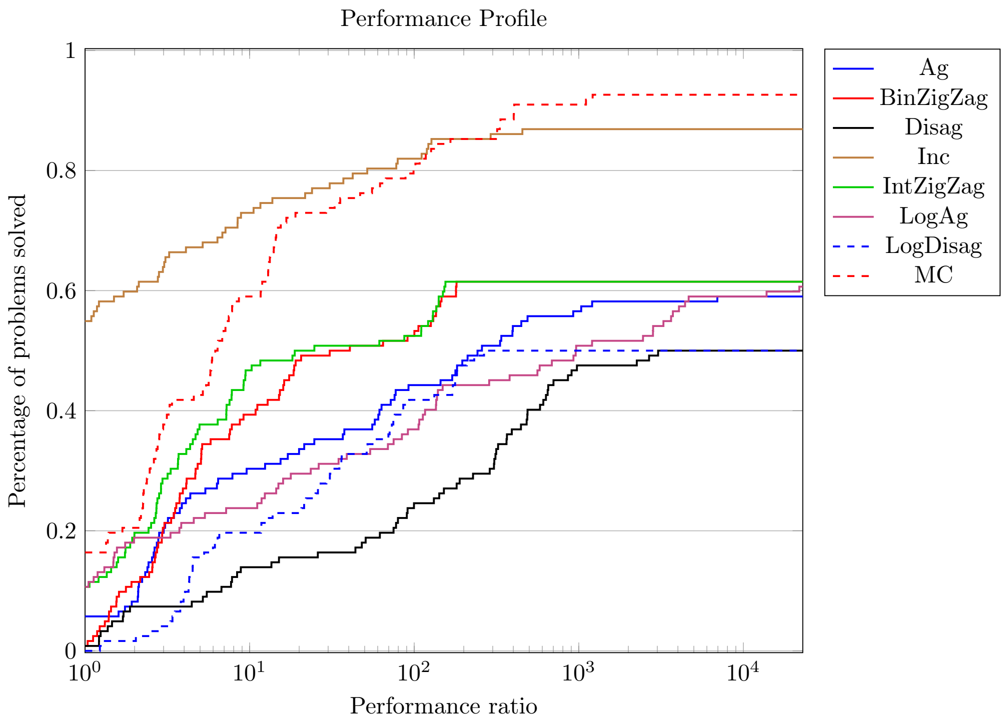
(13B) $\epsilon = 10^0$



(13C) $\epsilon = 10^{-2}$

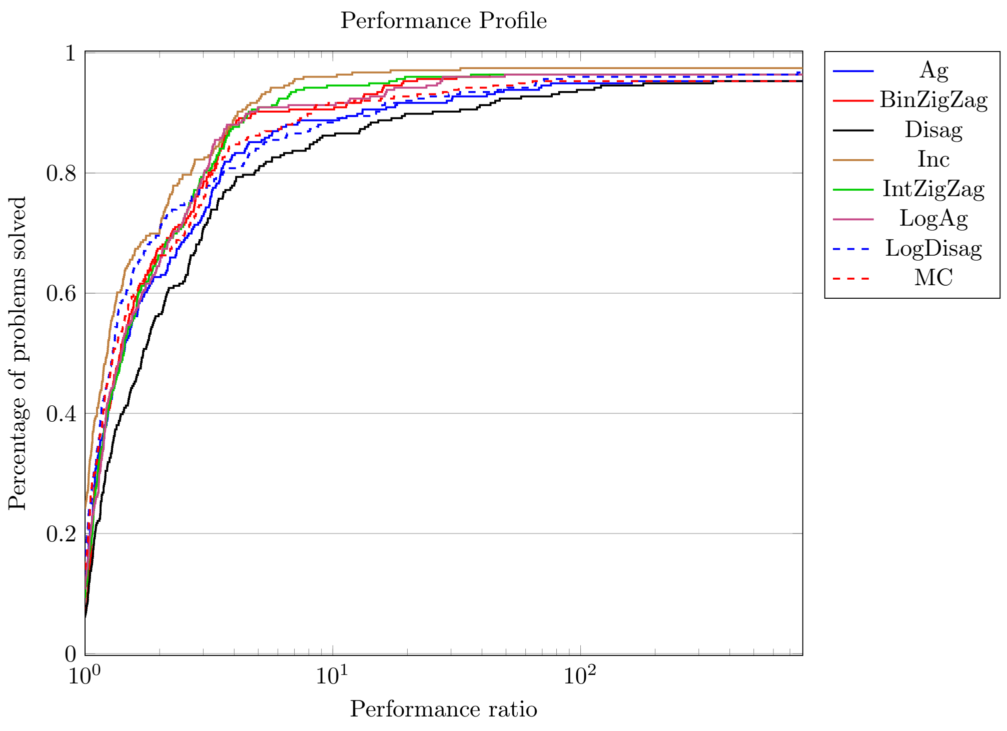


(13D) $\epsilon = 10^{-4}$

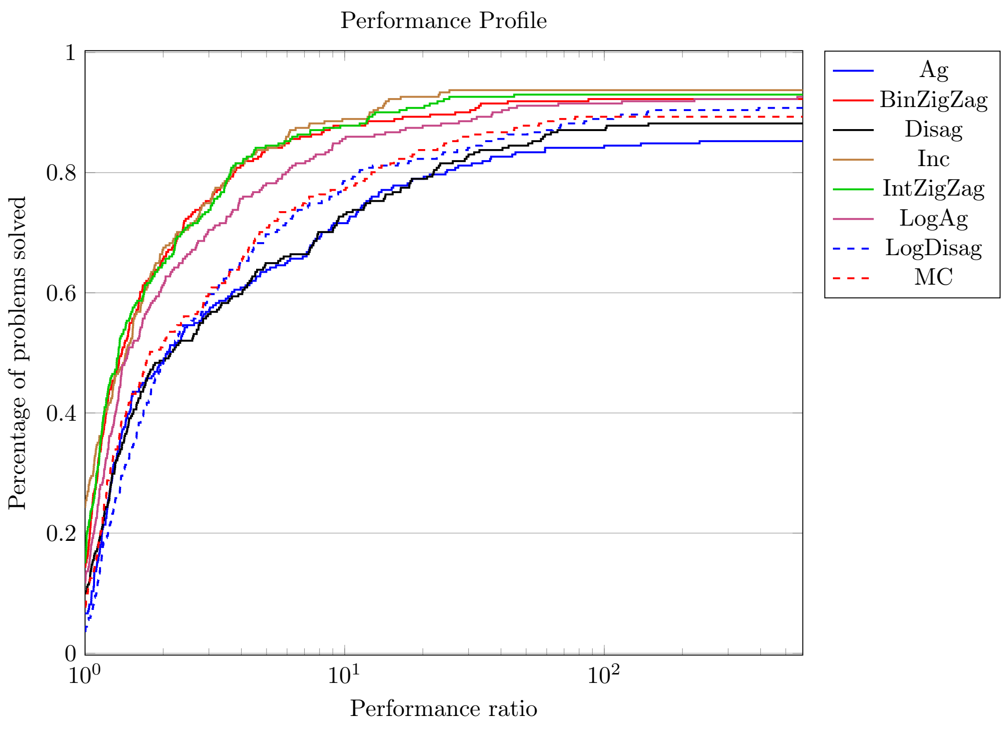


(13E) $\epsilon = 10^{-6}$

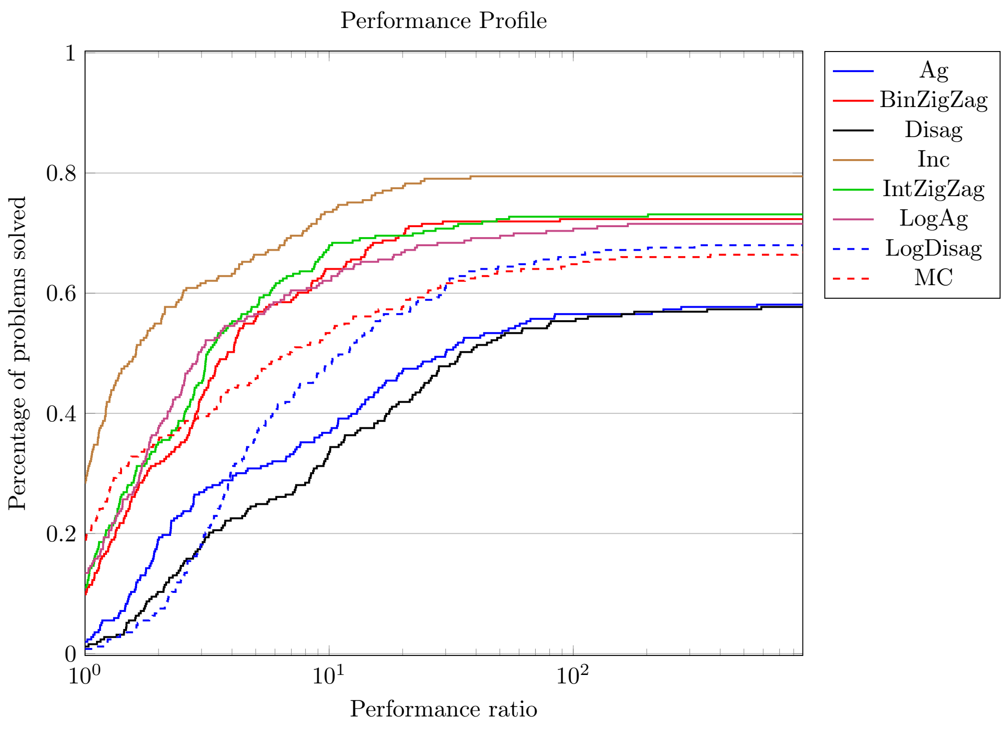
FIGURE 13. Performance profiles considering the time until a primal solution is found.



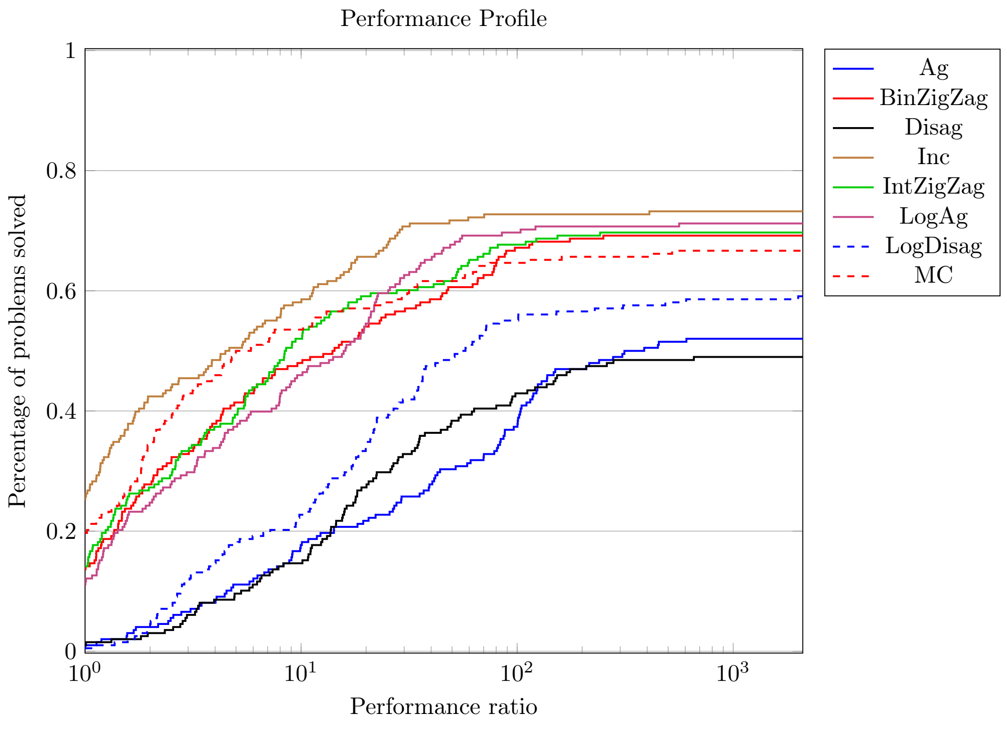
(14A) $\epsilon = 100$



(14B) $\epsilon = 1$



(14C) $\epsilon = 10^{-2}$



(14D) $\epsilon = 10^{-4}$

TABLE 11. Shifted geometric mean of the runtimes when the time limit is smaller.

Error bound	$\epsilon = 10^2$			$\epsilon = 10^0$		
Time limit	100s	1000s	10000s	100s	1000s	10000s
(Disag)	5.80	9.59	12.35	16.51	35.89	53.28
(LogDisag)	5.50	9.00	11.09	16.20	32.13	45.08
(Ag)	5.32	8.91	11.22	16.57	36.60	58.28
(LogAg)	4.92	8.05	9.78	13.42	25.98	35.87
(Inc)	4.66	7.68	9.28	13.38	25.05	33.34
(MC)	5.18	8.87	11.36	15.45	31.58	45.77
(BinZigZag)	4.80	7.90	9.75	12.84	23.68	32.27
(IntZigZag)	4.92	8.01	9.93	12.75	23.53	31.42

Error bound	$\epsilon = 10^{-2}$			$\epsilon = 10^{-4}$		
Time limit	100s	1000s	10000s	100s	1000s	10000s
(Disag)	43.23	174.32	530.03	52.12	231.03	830.62
(LogDisag)	43.61	147.96	356.86	51.78	227.33	654.58
(Ag)	42.24	171.69	517.11	55.48	272.14	980.37
(LogAg)	37.77	111.15	251.99	46.55	181.88	413.56
(Inc)	34.51	106.69	217.44	35.43	127.19	299.46
(MC)	36.60	127.29	344.90	37.03	124.62	318.77
(BinZigZag)	36.91	113.34	248.81	42.79	178.65	429.61
(IntZigZag)	36.92	112.13	243.39	42.15	157.64	371.29

Error bound	$\epsilon = 10^{-6}$		
Time limit	100s	1000s	10000s
(Disag)	70.91	370.48	1382.15
(LogDisag)	66.32	305.45	1021.40
(Ag)	73.83	427.68	1611.82
(LogAg)	51.06	189.23	491.73
(Inc)	39.67	105.94	207.93
(MC)	53.22	154.46	282.39
(BinZigZag)	49.11	179.97	496.05
(IntZigZag)	49.09	166.62	438.72

TABLE 12. Shifted geometric mean of the runtimes when the time limit is smaller, considering the subset of benchmark instances, where all reformulations found a feasible solution.

Error bound	$\epsilon = 10^2$			$\epsilon = 10^0$		
Time limit	100s	1000s	10000s	100s	1000s	10000s
(Disag)	5.24	8.21	9.97	13.67	25.87	32.43
(LogDisag)	4.98	7.83	9.24	13.37	22.93	27.96
(Ag)	4.77	7.57	8.94	13.73	26.47	36.44
(LogAg)	4.43	6.97	8.00	10.71	18.06	21.79
(Inc)	4.13	6.48	7.52	10.67	16.95	19.27
(MC)	4.63	7.53	9.07	12.65	22.25	27.41
(BinZigZag)	4.32	6.81	7.97	10.17	16.24	19.31
(IntZigZag)	4.43	6.92	8.13	10.06	16.01	18.82

Error bound	$\epsilon = 10^{-2}$			$\epsilon = 10^{-4}$		
Time limit	100s	1000s	10000s	100s	1000s	10000s
(Disag)	25.90	63.69	103.27	25.48	49.19	67.16
(LogDisag)	26.31	52.05	76.86	25.10	48.53	56.12
(Ag)	24.88	61.90	106.73	29.38	70.97	103.53
(LogAg)	20.66	36.24	51.78	19.46	32.35	34.80
(Inc)	17.27	29.73	40.72	9.99	15.91	17.83
(MC)	19.50	38.55	58.16	12.77	20.81	26.81
(BinZigZag)	19.86	36.52	51.87	15.71	30.42	34.52
(IntZigZag)	19.80	36.27	51.96	15.91	27.26	31.29

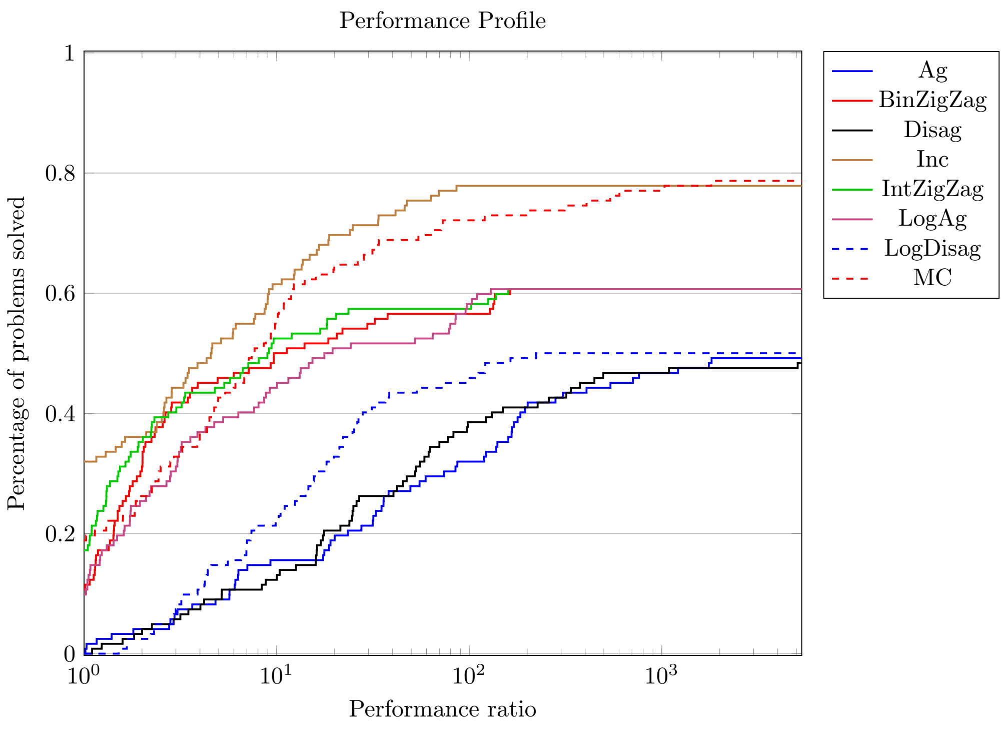
Error bound	$\epsilon = 10^{-6}$		
Time limit	100s	1000s	10000s
(Disag)	45.02	114.37	153.65
(LogDisag)	37.43	65.88	65.88
(Ag)	48.86	149.92	225.51
(LogAg)	18.77	22.83	22.83
(Inc)	23.48	32.31	33.19
(MC)	32.27	65.67	77.53
(BinZigZag)	17.63	22.40	22.40
(IntZigZag)	17.90	20.10	20.10

Table 13. Shifted geometric mean of the runtimes when the time limit is smaller, considering the subset of benchmark instances, where all reformulations found an optimal solution.

Error bound	$\epsilon = 10^2$			$\epsilon = 10^0$		
Time limit	100s	1000s	10000s	100s	1000s	10000s
(Disag)	4.24	5.87	6.16	10.40	16.95	17.98
(LogDisag)	4.02	5.69	5.93	10.14	14.58	15.47
(Ag)	3.79	5.29	5.39	10.33	16.15	17.35
(LogAg)	3.52	4.97	5.10	7.78	11.09	11.76
(Inc)	3.18	4.54	4.66	7.71	10.95	11.60
(MC)	3.65	5.25	5.44	9.48	14.06	14.98
(BinZigZag)	3.39	4.78	4.93	7.34	10.32	10.93
(IntZigZag)	3.47	4.84	4.97	7.31	10.22	10.77

Error bound	$\epsilon = 10^{-2}$			$\epsilon = 10^{-4}$		
Time limit	100s	1000s	10000s	100s	1000s	10000s
(Disag)	17.39	31.24	35.23	20.75	32.43	36.76
(LogDisag)	17.96	26.00	28.31	20.45	33.05	34.80
(Ag)	16.11	27.18	29.68	23.78	45.53	52.14
(LogAg)	13.22	17.44	18.46	15.03	21.98	22.21
(Inc)	10.27	12.69	12.81	6.24	7.84	7.84
(MC)	11.55	15.06	15.68	8.73	10.68	11.06
(BinZigZag)	12.87	17.90	18.78	11.54	21.12	22.72
(IntZigZag)	12.58	17.23	18.43	11.83	18.81	19.98

Error bound	$\epsilon = 10^{-6}$		
Time limit	100s	1000s	10000s
(Disag)	41.19	96.32	119.07
(LogDisag)	35.70	62.14	62.14
(Ag)	45.15	125.78	161.97
(LogAg)	17.57	21.70	21.70
(Inc)	20.02	26.35	27.19
(MC)	28.27	53.41	60.71
(BinZigZag)	16.59	21.33	21.33
(IntZigZag)	16.04	18.32	18.32



(14E) $\epsilon = 10^{-6}$

FIGURE 14. Performance profiles considering the time until an optimal solution is found.

Acknowledgments

The authors gratefully acknowledge the scientific support and HPC resources provided by the Erlangen National High Performance Computing Center (NHR@FAU) of the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU). The hardware is funded by the German Research Foundation (DFG). This work has been done within the joint project "TrinkXtrem" funded by the Federal Ministry of Education and Research (BMBF) under the project number 02WEE1625B in the funding "Wasser-Extremereignisse" (WaX) of the Federal Program "Wasser:N" and as part of the announcement "Artificial Intelligence in Civil Security Research II" of the BMBF within the program "Research for Civil Security" of the Federal Government.

References

Aigner, Kevin-Martin, Robert Burlacu, Frauke Liers, and Alexander Martin (2023). “Solving AC Optimal Power Flow with Discrete Decisions to Global Optimality”. In: *INFORMS Journal on Computing* 35.2, pp. 458–474. DOI: 10.1287/ijoc.2023.1270.

Balakrishnan, Anantharam and Stephen C. Graves (1989). “A composite algorithm for a concave-cost network flow problem”. In: *Networks* 19.2, pp. 175–202. DOI: 10.1002/net.3230190202.

Bärmann, Andreas, Robert Burlacu, Lukas Hager, and Thomas Kleinert (2022). “On piecewise linear approximations of bilinear terms: structural comparison of univariate and bivariate mixed-integer programming formulations”. In: *Journal of Global Optimization* 85.4, pp. 789–819. DOI: 10.1007/S10898-022-01243-y.

Beach, Benjamin, Robert Hildebrand, and Joey Huchette (2022). “Compact mixed-integer programming formulations in quadratic optimization”. In: *Journal of Global Optimization* 84.4, pp. 869–912. DOI: 10.1007/s10898-022-01184-6.

Belotti, Pietro, Sonia Cafieri, Jon Lee, and Leo Liberti (2010). “Feasibility-Based Bounds Tightening via Fixed Points”. In: *Combinatorial Optimization and Applications*. Ed. by Weili Wu and Ovidiu Daescu. Vol. 6508. Springer Berlin Heidelberg, pp. 65–76. DOI: 10.1007/978-3-642-17458-2_7.

Burlacu, Robert, Björn Geißler, and Lars Schewe (2019). “Solving mixed-integer nonlinear programmes using adaptively refined mixed-integer linear programmes”. In: *Optimization Methods and Software* 35, pp. 37–64. DOI: 10.1080/10556788.2018.1556661.

Bussieck, Michael R., Arne Stolbjerg Drud, and Alexander Meeraus (2003). “MINLPLib—A Collection of Test Models for Mixed-Integer Nonlinear Programming”. In: *INFORMS Journal on Computing* 15.1, pp. 114–119. DOI: 10.1287/ijoc.15.1.114.15159.

Bynum, Michael L, Gabriel A Hackebeil, William E Hart, Carl D Laird, Bethany L Nicholson, John D Siirola, Jean-Paul Watson, David L Woodruff, et al. (2021). *Pyomo-optimization modeling in python*. Vol. 67. Springer. DOI: 10.1007/978-3-030-68928-5.

Correa-Posada, Carlos M. and Pedro Sánchez-Martín (2014). "Gas Network Optimization: A comparison of Piecewise Linear Models". URL: http://www.optimization-online.org/DB_HTML/2014/10/4580.html.

Croxton, Keely L., Bernard Gendron, and Thomas L. Magnanti (2003). "A Comparison of Mixed-Integer Programming Models for Nonconvex Piecewise Linear Cost Minimization Problems". In: *Management Science* 49.9, pp. 1268–1273. DOI: 10.1287/mnsc.49.9.1268.16570.

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

Fourer, Robert, Jun Ma, and Kipp Martin (2010). "OSiL: An instance language for optimization". In: *Computational optimization and applications* 45.1, pp. 181–203. DOI: 10.1007/s10589-008-9169-6.

Geißler, Björn, Alexander Martin, Antonio Morsi, and Lars Schewe (2012). "Using Piecewise Linear Functions for Solving MINLPs". In: *Mixed Integer Nonlinear Programming*. Ed. by Jon Lee and Sven Leyffer. Vol. 154. The IMA Volumes in Mathematics and its Applications. Springer New York, pp. 287–314. DOI: 10.1007/978-1-4614-1927-3_10.

Gugat, Martin, Günter Leugering, Alexander Martin, Martin Schmidt, Mathias Sirvent, and David Wintergerst (2018). "Towards simulation based mixed-integer optimization with differential equations". In: *Networks*. DOI: 10.1002/net.21812.

Gurobi Optimization, LLC (2023). *Gurobi Optimizer Reference Manual*. URL: https://www.gurobi.com.

Hart, William E, Jean-Paul Watson, and David L Woodruff (2011). "Pyomo: modeling and solving mathematical programs in Python". In: *Mathematical Programming Computation* 3.3, pp. 219–260. DOI: 10.1007/s12532-011-0026-8.

Hasan, M. M. Faruque and I.A. Karimi (2010). "Piecewise linear relaxation of bilinear programs using bivariate partitioning". In: *AIChE Journal* 56.7, pp. 1880–1893. DOI: 10.1002/aic.12109.

Huchette, Joey and Juan Pablo Vielma (2022). "Nonconvex Piecewise Linear Functions: Advanced Formulations and Simple Modeling Tools". In: *Operations Research* 71.5, pp. 1835–1856. DOI: 10.1287/opre.2019.1973.

Lee, Jon and Dan Wilson (2001). "Polyhedral methods for piecewise-linear functions. I. The lambda method". In: *Discrete Appl. Math.* 108.3, pp. 269–285. DOI: 10.1016/S0166-218x(00)00216-x.

Link, Moritz and Stefan Volkwein (2023). "Adaptive piecewise linear relaxations for enclosure computations for nonconvex multiobjective mixed-integer quadratically constrained programs". In: *Journal of Global Optimization* 87.1, pp. 97–132. DOI: 10.1007/s10898-023-01309-5.

Lundell, Andreas, Anders Skjäl, and Tapio Westerlund (2013). "A reformulation framework for global optimization". In: *Journal of Global Optimization* 57.1, pp. 115–141. DOI: 10.1007/s10898-012-9877-4.

Markowitz, Harry M and Alan S Manne (1957). "On the Solution of Discrete Programming Problems". In: *Econometrica* 25.1, pp. 84–110. DOI: 10.2307/1907744.

Martin, Alexander, Markus Möller, and Susanne Moritz (2006). “Mixed Integer Models for the Stationary Case of Gas Network Optimization”. In: *Mathematical Programming* 105.2, pp. 563–582. DOI: 10.1007/s10107-005-0665-5.

McCormick, Garth P (1976). “Computability of global solutions to factorable nonconvex programs: Part I—Convex underestimating problems”. In: *Mathematical programming* 10.1, pp. 147–175. DOI: 10.1007/bf01580665.

Misener, R. and C. A. Floudas (2010). “Piecewise-Linear Approximations of Multidimensional Functions”. In: *Journal of Optimization Theory and Applications* 145.1, pp. 120–147. DOI: 10.1007/s10957-009-9626-0.

Morsi, Antonio (2013). “Solving MINLPs on Loosely-Coupled Networks with Applications in Water and Gas Network Optimization”. PhD thesis. Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU).

Padberg, Manfred (2000). “Approximating separable nonlinear functions via mixed zero-one programs”. In: *Operations Research Letters* 27.1, pp. 1–5. DOI: 10.1016/s0167-6377(00)00028-6.

Rebennack, Steffen and Josef Kallrath (2015a). “Continuous piecewise linear delta-approximations for bivariate and multivariate functions”. In: *Journal of Optimization Theory and Applications* 167.1, pp. 102–117. DOI: 10.1007/s10957-014-0688-2.

– (2015b). “Continuous piecewise linear delta-approximations for univariate functions: computing minimal breakpoint systems”. In: *Journal of Optimization Theory and Applications* 167.2, pp. 617–643. DOI: 10.1007/s10957-014-0687-3.

Rebennack, Steffen and Vitaliy Krasko (2020). “Piecewise Linear Function Fitting via Mixed-Integer Linear Programming”. In: *INFORMS Journal on Computing* 32.2, pp. 507–530. DOI: 10.1287/ijoc.2019.0890.

Rovatti, Ricardo, Claudia D’Ambrosio, Andrea Lodi, and Silvano Martello (2014). “Optimistic MILP modeling of non-linear optimization problems”. In: *European Journal of Operational Research* 239.1, pp. 32–45. DOI: 10.1016/j.ejor.2014.03.020.

Sherali, Hanif D. (2001). “On mixed-integer zero-one representations for separable lower-semicontinuous piecewise-linear functions”. In: *Operations Research Letters* 28.4, pp. 155–160. DOI: 10.1016/s0167-6377(01)00063-3.

Siqueira, Abel Soares, Raniere Gaia Costa da Silva, and Luiz-Rafael Santos (2016). “Perprof-py: A Python Package for Performance Profile of Mathematical Optimization Software”. In: *Journal of Open Research Software* 4.1, p. 12. DOI: 10.5334/jors.81.

Vielma, Juan Pablo (2015). “Mixed Integer Linear Programming Formulation Techniques”. In: *SIAM Review* 57.1, pp. 3–57. DOI: 10.1137/130915303.

Vielma, Juan Pablo, Shabbir Ahmed, and George L Nemhauser (2010). “Mixed-Integer Models for Nonseparable Piecewise-Linear Optimization: Unifying Framework and Extensions”. In: *Operations Research* 58.2, pp. 303–315. DOI: 10.1287/opre.1090.0721.