BENCHMARKING PIECEWISE LINEAR REFORMULATIONS FOR MINLPs: A COMPUTATIONAL STUDY BASED ON THE OPEN-SOURCE FRAMEWORK PWL-T-REX

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

[1] *Fraunhofer Institute for Integrated Circuits IIS, Nordostpark 84, D-90411 Nürnberg, Germany*

[2] *University of Technology Nuremberg, Ulmenstr. 52h, D-90443 Nürnberg, Germany*

[*] *Corresponding author,* kristin.braun@iis.fraunhofer.de

Abstract. Solving mixed-integer nonlinear problems by means of piecewise linear relaxations has emerged as 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 implement a framework that reformulates mixed-integer nonlinear problems to eight commonly used different mixed-integer representations for piecewise linear relaxations where the reformulations can be compared in terms of behavior and runtime to determine which method to apply in practice. We use expression trees to reformulate all nonlinearities to one-dimensional functions and then compute a set of interpolation breakpoints for each function based on a given maximum error. This framework is made publicly available, see [7].

Further, we conduct an analysis on a benchmark set created from the MINLPLIB consisting of over 750 instances. It includes a comprehensive comparison of the number of problems solved, runtimes, and optimality gaps.

Keywords: Piecewise linear relaxations, mixed-integer nonlinear programming, mixed-integer programming, discrete optimization

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.

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 [27, 15, 30, 25, 16, 8, 1, 24, 4]. 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 (MILP) 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 [34, 33] and [35]. This is supplemented by the construction of optimal polynomial relaxations of one-dimensional functions in [30]. The construction of optimal breakpoints for quadratic functions is presented in [32]. Specific approximation techniques for general nonlinear functions with dimensions smaller than three are proposed in [29]. The number of simplices in the approximation grows exponentially with the function's dimension. In this regard, we refer to the method of [36], 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.

This article covers a variety of different approaches existing for modeling PWL functions as a MILP: the disaggregated (as in [12, 37]) and aggregated convex combination models (as in [23, 31]) as well as their logarithmic variants presented by [40], the classical incremental method of [26], and the multiple choice model as in [2] and again [12]. 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 [20]. In [21], these methods are clustered and extensively tested on transportation problems. For more details on the theoretical foundations of MILP models for PWL functions and general mixed-integer formulation techniques, we refer to the extensive survey by [39].

To the best of our knowledge, no comprehensive study has yet been conducted that demonstrates the performance of these MILP models on a wide and general range of MINLP problems. In most cases, some parts of the mentioned MILP models are compared only on some instances of very specific problems; see for instance [11, 19, 21, 32]. Therefore, it becomes difficult to decide in general, which MILP modeling is preferable, or if there is a best modeling at all. To this end, we perform an extensive computational study on over 750 instances from the MINLPLIB, cf. [9]. Using expression trees and the results from [3], we reformulate all MINLP instances to equivalent models that consist of only one-dimensional nonlinear functions. Based on these reformulations, we compare various MILP relaxations corresponding to the different PWL models. Overall, our study shows that the incremental

TABLE 1. Overview of PWL models.

Full name	**Short name**	**Presented in**	**Reference(s)**
Disaggregated	(Disag)	Section 3.1	[40], cf. refs
Log. Disaggregated	(LogDisag)	Section 3.1	[40]
Aggregated	(Ag)	Section 3.2	[40], cf. refs
Log. Aggregated	(LogAg)	Section 3.2	[40]
Binary Zig-Zag	(BinZigZag)	Section 3.2	[20]
Integer Zig-Zag	(IntZigZag)	Section 3.2	[20]
Incremental	(Inc)	Section 3.3	[26]
Multiple Choice	(MC)	Section 3.4	[22]

method, although the oldest approach of all, generally performs best when MINLPs are solved by PWL relaxations that are modeled as MILPs, while the very recently introduced logarithmic integer Zig-Zag model by [20] is a reasonable alternative for very high-accuracy PWL relaxations. While the multiple choice method is rarely recommended in the literature, our findings offer compelling evidence in its efficiency in finding primal feasible solutions.
Contribution. In this work, we conduct a large-scale computational study comparing eight different MILP formulations based on PWL relaxations. Our key contributions include:

(1) A framework for generating and solving MINLPs via piecewise linear relaxations, with selectable MILP reformulations and error bounds. Our framework is made publicly available as an open-source implementation on GitHub, see [7].
(2) A detailed performance evaluation on over 750 instances from MINLPLIB, analyzing problem solvability, runtime, and optimality gaps.
(3) The creation and validation of a branching scheme for the logarithmic aggregated convex combination method, which is used in our computational study.

Structure. 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 MILP models for PWL relaxations, see Table 1. After describing our framework in Section 4, we present the computational study in Section 5 that illustrates the practicability of the various MILP 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:

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

where $A \in \mathbb{R}^{m \times (p+q)}, b \in \mathbb{R}^m, c \in \mathbb{R}^{p+q}$ and $k, q, p, m \in \mathbb{N}$. The variables x can be bounded from below and above by $l \in (\mathbb{R} \cup \{-\infty\})^{q+p}$ and $u \in (\mathbb{R} \cup \{\infty\})^{q+p}$. The inequalities $Ax \leq b$ denote the linear constraints, while the nonlinear constraints are denoted by $\mathcal{F} := \{f_1, \dots, f_k\}$. Each function $f_i \in \mathcal{F}$ is a nonlinear, real-valued function $f_i \colon D_{f_i} \to \mathbb{R}$ where $D_{f_i} \subset \mathbb{R}^{q+p}$ is the domain of f_i. Equality constraints, i.e., constraints of type $f_i(x) = 0$, are inherently contained in the description (P) by simply adding $f_i(x) \leq 0$ and $-f_i(x) \leq 0$. Further, we are not restricted to a linear objective function $c^\top x$, since we can include any nonlinear objective function $f_c \colon D_{f_c} \to \mathbb{R}$ by substituting $f_c(x)$ with a variable $y \in \mathbb{R}$ and adding $f_c(x) \leq y$ as a constraint to (P). 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.

In our work, we assume that each variable in (P) has lower and upper bounds $l, u \in \mathbb{R}$. Additionally, we enforce that nonlinear functions f are bounded through the choice of D_f. Further, our work is restricted to continuous functions f. By this, the domain D_f is defined as a d-dimensional box with $d \leq q + p$, whose edges are parallel to the coordinate axes, making it a compact set.

To solve problems of type (P) with box-constrained domains, we employ piecewise linear (PWL) relaxations. For small values of d, this approach presents a viable alternative to spatial branching. By using PWL relaxations, we permit solutions to be slightly infeasible within a specified error bound, effectively controlling the approximation quality. However, MINLPs are typically solved only within given feasibility and optimality tolerances, so this is not a significant limitation.

A common approach to PWL relaxations is to first construct a PWL approximation of f by interpolating the function over a d-dimensional simplex. The corresponding approximation error is then incorporated into the relaxation to ensure validity. Relaxations are preferable to approximations because they preserve all feasible solutions, thereby guaranteeing valid bounds, whereas approximations may exclude feasible solutions, leading to missing upper or lower bounds. To achieve this, we first construct a triangulation of D_f. However, the number of simplices in a triangulation of a d-dimensional box grows exponentially with d for a fixed approximation quality. As a result, this method is only practical for low-dimensional cases.

For a wide range of practically interesting MINLPs, the nonlinear functions are factorable, and can be represented by the following functions:

$$\begin{aligned}&f(x_1,x_2)=x_1\cdot x_2,\ f(x_1,x_2)=\frac{x_1}{x_2},\ f(x)=\ln x,\ f(x)=\log_{10}x,\\&f(x)=x^a \text{ (including } x^{-1} \text{ and } \sqrt{x}),\ f(x)=a^x \text{ (including } e^x),\\&f(x)=\sin x,\ f(x)=\cos x,\ f(x)=\tanh x,\ f(x)=|x|.\end{aligned}\tag{1}$$

as described by [5]. The excessive use of this leads to an equivalent formulation of the MINLP problem that contains only univariate nonlinearities and, if necessary, the coupling bivariate function $f(x_1,x_2)=x_1x_2$. Following the results of [3], we can further reformulate $f(x_1,x_2)$ to a 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 $p\in\mathbb{R}$. 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 McCormick relaxations from [28]:

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

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

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

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

where x_1^-, x_1^+, x_2^-, and x_2^+ are the lower and upper bounds of x_1 and x_2, respectively. We use this reformulation as it tightens the relaxation, leading to improved solver performance.

This gives an equivalent reformulation of the MINLP that consists solely of one-dimensional nonlinearities. Hence, in the following, we consider only one-dimensional MILP models for PWL relaxations of nonlinear functions.

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 MILP models for PWL relaxations with different approximation errors.

3. One-dimensional MILP models for piecewise linear relaxations

In this chapter, we describe in more detail the MILP representations of the piecewise linear models that we use in this work. For all representations, we show the MILP 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, a binary variable $y_i \in \{0, 1\}$ is introduced 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

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

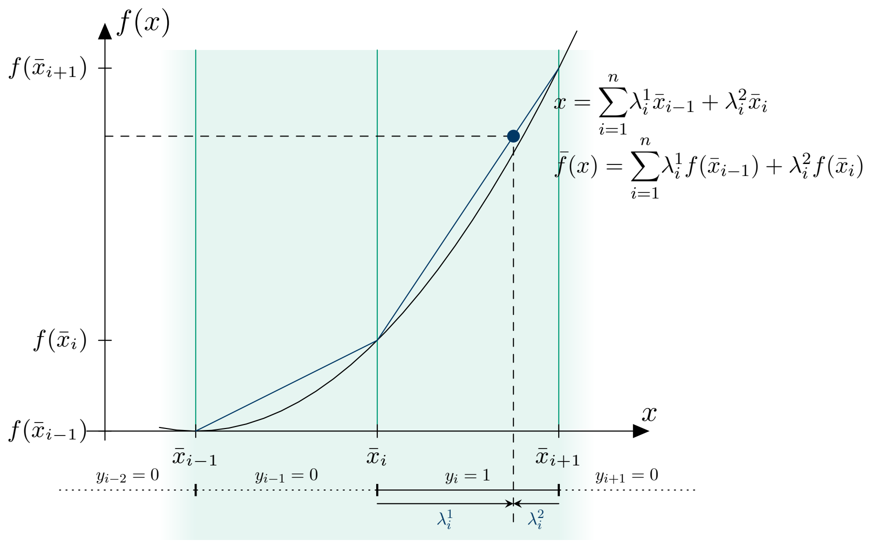


FIGURE 1. Representation of (Disag).

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, new binary variables y_l with $l \in \{1, ..., \lceil \log_2 n \rceil\}$ and a binary encoding $B : [n] \to \{0,1\}^{\lceil \log_2 n \rceil}$ are introduced. The binary encoding is not fixed; in our framework, we simply use the usual conversion from decimal to binary system. Using this encoding, one can define a branching scheme by

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

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, the previous model just needs to be adjusted 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, are aggregated. To this end, new continuous variables $\lambda_0, \dots, \lambda_n$ are introduced. 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, one only needs $n+1$ 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.

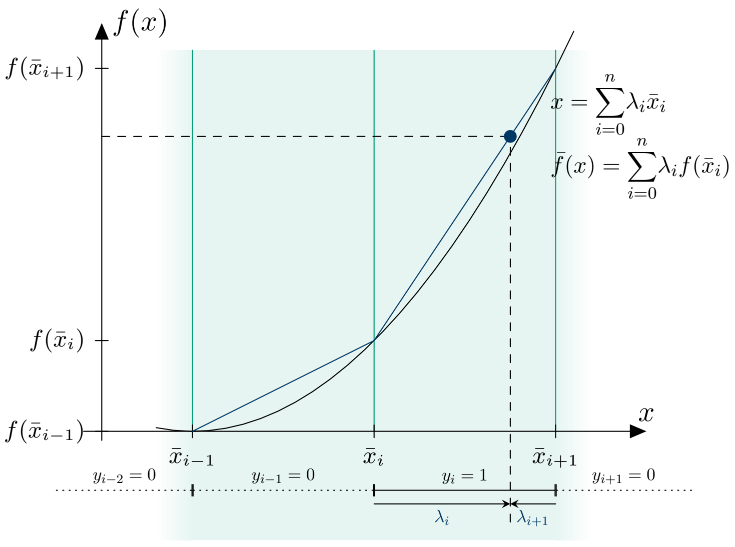


FIGURE 2. Representation of (Ag).

Model 3. Convex combination model

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

As before, one wants to reduce the number of binary variables in our model. For this, we present different approaches: On the one hand, a formulation using a binary branching scheme, and, on the other hand, two new formulations introduced by [20].

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 as in the aggregated version here, 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 $\left(\mathcal{L}^S, \mathcal{R}^S\right)$ with $\mathcal{L}^S := \left\{L_1^S, \dots, L_S^S\right\}$ and $\mathcal{R}^S := \left\{R_1^S, \dots, R_S^S\right\}$ for $S := \lceil \log_2 n \rceil$ such that for all $i \in \{1, \dots, n\}$ there exists a series $T^i = [T_1^i, \dots, T_S^i]$ with

$$\{i-1, i\} = \bigcap_{s=1}^{S} [n]_0 \setminus T_s^i, \tag{9}$$

where $T_s^i \in \left\{L_s^S, R_s^S\right\}$ and $[n]_0 := [n] \cup \{0\} = \{0, \dots, n\}$. We defined such a branching scheme for this work and describe it in the following.

Definition 4. For a given $S \geq 1$, our branching scheme $\left(\mathcal{L}^S, \mathcal{R}^S\right)$ is defined by

$$\begin{aligned} \mathcal{L}^S &:= \left\{L_1^S|_{[n]_0}, \dots, L_S^S|_{[n]_0}\right\} \text{ and} \\ \mathcal{R}^S &:= \left\{R_1^S|_{[n]_0}, \dots, R_S^S|_{[n]_0}\right\}, \end{aligned} \tag{10}$$

where

$$\begin{aligned} L_S^S &:= \left\{0, \dots, 2^{S-1} - 1\right\}, \\ R_S^S &:= \left\{2^{S-1} + 1, \dots, 2^S\right\}, \end{aligned} \tag{11}$$

and, for all $s \in [S-1]$,

$$\begin{aligned} L_s^S &:= \left(L_s^{S-1} \cup \left\{2^S - j : j \in L_s^{S-1}\right\}\right), \\ R_s^S &:= \left(R_s^{S-1} \cup \left\{2^S - j : j \in R_s^{S-1}\right\}\right). \end{aligned} \tag{12}$$

In (12), L_s^{S-1} and R_s^{S-1} are recursively calculated.

The following Lemma 5 shows that Definition 4 fulfills the necessary conditions.

Lemma 5. *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 $\left(\mathcal{L}^S, \mathcal{R}^S\right)$ for $S := \lceil \log_2(n) \rceil$ be given as in Definition 4. Then, for all segments $[\bar{x}_{i-1}, \bar{x}_i]$ with $1 \leq i \leq n$, there exists a series of sets $T^i := [T_1^i, \dots, T_S^i]$ with $T_s^i \in \left\{L_s^S, R_s^S\right\}$, $s \in [S]$, such that*

$$[n]_0 \setminus \{i-1, i\} = \bigcup_{s=1}^{S} T_s^i, \tag{13}$$

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 $T_1^1 := R_1^1$ and $T_1^2 := L_1^1$. Then, we have

$$\bigcup_{s=1}^{S} T_s^i = \begin{cases} R_1^1 = \{2\} = \bar{X}_2 \setminus \{0, 1\} & \text{if } i = 1, \\ L_1^1 = \{0\} = \bar{X}_2 \setminus \{1, 2\} & \text{if } i = 2. \end{cases} \tag{14}$$

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

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

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

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

Using (15), we can reformulate this to

$$\begin{aligned}\bigcup_{s=1}^{S-1} T_s^i &= \left(\bar{X}_{2^{S-1}} \setminus \{i-1, i\}\right) \cup \left(\left\{2^S - j : j \in \bar{X}_{2^{S-1}} \setminus \{i-1, i\}\right\}\right) \\ &= \left(\bar{X}_{2^{S-1}} \setminus \{i-1, i\}\right) \cup \left(\left\{2^{S-1}, \dots, 2^S\right\} \setminus \left\{2^S - i, 2^S - (i-1)\right\}\right) \\ &= \bar{X}_{2^S} \setminus \left\{i-1, i, 2^S - i, 2^S - (i-1)\right\} \\ &= [n]_0 \setminus \left\{i-1, i, 2^S - i, 2^S - (i-1)\right\}.\end{aligned} \tag{17}$$

Now, we need to prove that, using the new sets L_S^S and R_S^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 $T_S^i = R_S^S$. Then,

$$\begin{aligned}\bigcup_{s=1}^{S} T_s^i = T_S^i \cup \bigcup_{s=1}^{S-1} T_s^i &= R_S^S \cup \left([n]_0 \setminus \left\{i-1, i, 2^S - i, 2^S - (i-1)\right\}\right) \\ &= \left\{2^{S-1}+1, \dots, n\right\} \cup \left([n]_0 \setminus \left\{i-1, i, 2^S - i, 2^S - (i-1)\right\}\right) \\ &= [n]_0 \setminus \{i-1, i\}.\end{aligned} \tag{18}$$

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

$$\begin{aligned}\bigcup_{s=1}^{S} T_s^i = T_S^i \cup \bigcup_{s=1}^{S-1} T_s^i &= L_S^S \cup \left([n]_0 \setminus \left\{i-1, i, 2^S - i, 2^S - (i-1)\right\}\right) \\ &= \left\{0, \dots 2^{S-1} - 1\right\} \cup \left([n]_0 \setminus \left\{i-1, i, 2^S - i, 2^S - (i-1)\right\}\right) \\ &= [n]_0 \setminus \{i-1, i\}.\end{aligned}$$
□

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

Model 6. 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 [20]. For these reformulations, one needs the encoding C^r that is defined recursively as

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

where $d = 2^k$, $C_d^k \in \mathbb{R}^k$ is the d-th row of C^k and $0^d, 1^d \in \mathbb{R}^d$ contain only zeroes or ones, respectively. 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:

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

$$
\sum_{v=0}^{n} C_{v,k}^r \lambda_v \leq y_k \leq \sum_{v=0}^{n} C_{v+1,k}^r \lambda_v. \tag{21}
$$

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 [20].

Model 7. 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 $r = \lceil \log_2 (n-1) \rceil$.

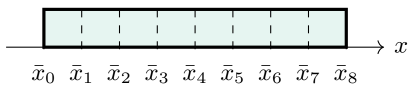
Model 8. General Integer Zig-Zag Formulation

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

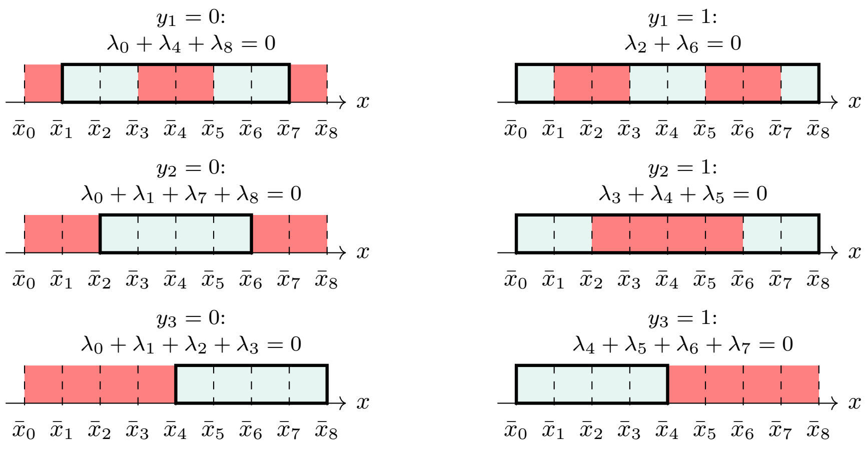
with $r = \lceil \log_2 (n-1) \rceil$.

(LogAg) and the Zig-Zag methods (BinZigZag) and (IntZigZag) have different branching behaviors. For an efficient solving of MILPs, it is important to quickly find tight dual bounds that are obtained by solving 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 [20], 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


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)

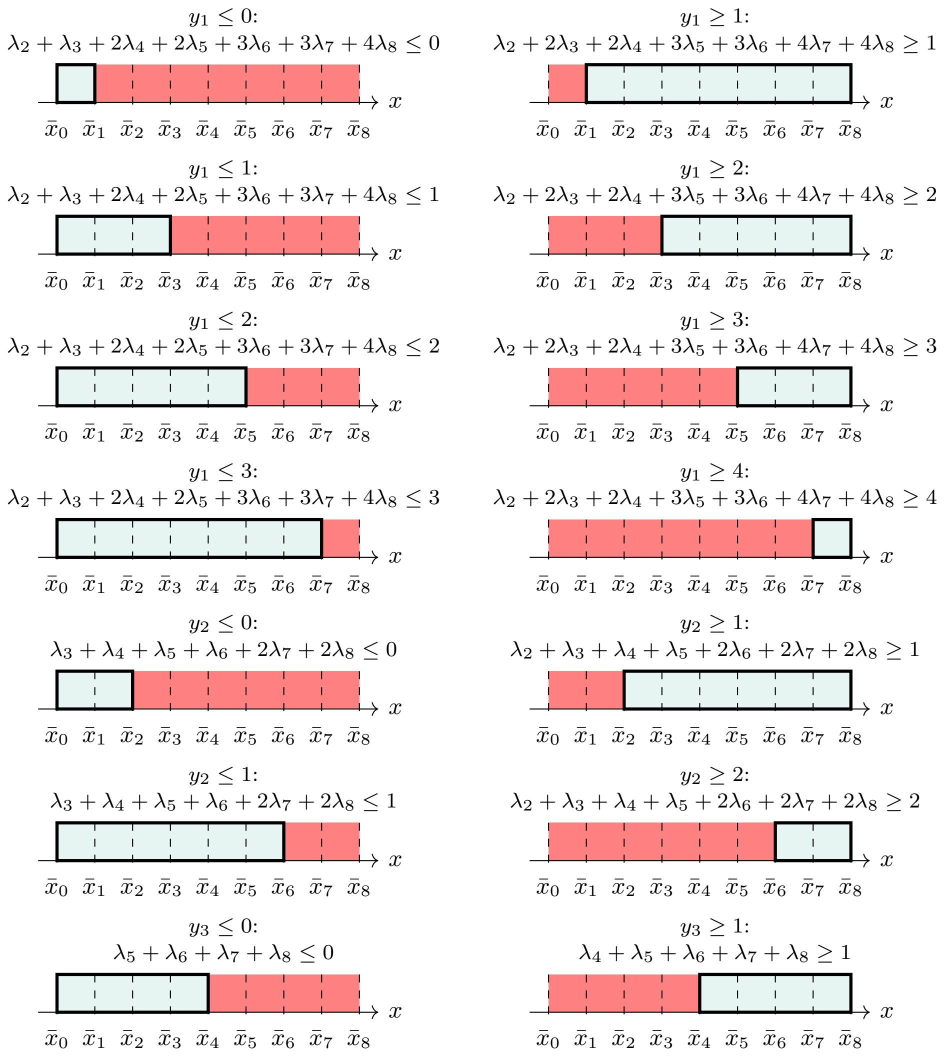
hull of the feasible set is outlined by a thick rectangle. Here, the convex hull is equivalent to the full domain.

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, 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 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

$$\lambda_0 + \lambda_1 + \lambda_2 + \lambda_3 \le y_3, \tag{22a}$$
$$\lambda_5 + \lambda_6 + \lambda_7 + \lambda_8 \le 1 - y_3, \tag{22b}$$



$y_1 \leq 0$:
$\lambda_2 + \lambda_3 + 2\lambda_4 + 2\lambda_5 + 3\lambda_6 + 3\lambda_7 + 4\lambda_8 \leq 0$
$y_1 \geq 1$:
$\lambda_2 + 2\lambda_3 + 2\lambda_4 + 3\lambda_5 + 3\lambda_6 + 4\lambda_7 + 4\lambda_8 \geq 1$
$y_1 \leq 1$:
$\lambda_2 + \lambda_3 + 2\lambda_4 + 2\lambda_5 + 3\lambda_6 + 3\lambda_7 + 4\lambda_8 \leq 1$
$y_1 \geq 2$:
$\lambda_2 + 2\lambda_3 + 2\lambda_4 + 3\lambda_5 + 3\lambda_6 + 4\lambda_7 + 4\lambda_8 \geq 2$
$y_1 \leq 2$:
$\lambda_2 + \lambda_3 + 2\lambda_4 + 2\lambda_5 + 3\lambda_6 + 3\lambda_7 + 4\lambda_8 \leq 2$
$y_1 \geq 3$:
$\lambda_2 + 2\lambda_3 + 2\lambda_4 + 3\lambda_5 + 3\lambda_6 + 4\lambda_7 + 4\lambda_8 \geq 3$
$y_1 \leq 3$:
$\lambda_2 + \lambda_3 + 2\lambda_4 + 2\lambda_5 + 3\lambda_6 + 3\lambda_7 + 4\lambda_8 \leq 3$
$y_1 \geq 4$:
$\lambda_2 + 2\lambda_3 + 2\lambda_4 + 3\lambda_5 + 3\lambda_6 + 4\lambda_7 + 4\lambda_8 \geq 4$
$y_2 \leq 0$:
$\lambda_3 + \lambda_4 + \lambda_5 + \lambda_6 + 2\lambda_7 + 2\lambda_8 \leq 0$
$y_2 \geq 1$:
$\lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 + 2\lambda_6 + 2\lambda_7 + 2\lambda_8 \geq 1$
$y_2 \leq 1$:
$\lambda_3 + \lambda_4 + \lambda_5 + \lambda_6 + 2\lambda_7 + 2\lambda_8 \leq 1$
$y_2 \geq 2$:
$\lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 + 2\lambda_6 + 2\lambda_7 + 2\lambda_8 \geq 2$
$y_3 \leq 0$:
$\lambda_5 + \lambda_6 + \lambda_7 + \lambda_8 \leq 0$
$y_3 \geq 1$:
$\lambda_4 + \lambda_5 + \lambda_6 + \lambda_7 + \lambda_8 \geq 1$
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.

$$\lambda_0 + \lambda_1 + \lambda_7 + \lambda_8 \leq y_2, \tag{22c}$$

$$\lambda_3 + \lambda_4 + \lambda_5 \leq 1 - y_2, \tag{22d}$$

$$\lambda_0 + \lambda_4 + \lambda_8 \leq y_1, \tag{22e}$$

$$\lambda_2 + \lambda_6 \leq 1 - y_1. \tag{22f}$$

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

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

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 $y_1 \le 2$ and $y_1 \ge 3$, given in the third line of Figure 3c. From (23a), we obtain

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

for the left branch with $y_1 \le 2$. 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,

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

emerges for $y_1 \ge 3$. Here, using any variable of $\lambda_0, \lambda_1, \dots, \lambda_5$ 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 [26]. It uses a linear number of binary variables, but a different behavior compared to the convex combination formulations. There are $n-1$ binary variables $y_1, \dots, y_{n-1}$ and a value $y_j = 1$ enforces that any segment i with $i \ge j$ is used. The other way around, one 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 advantage of setting all leftmost binary 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

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

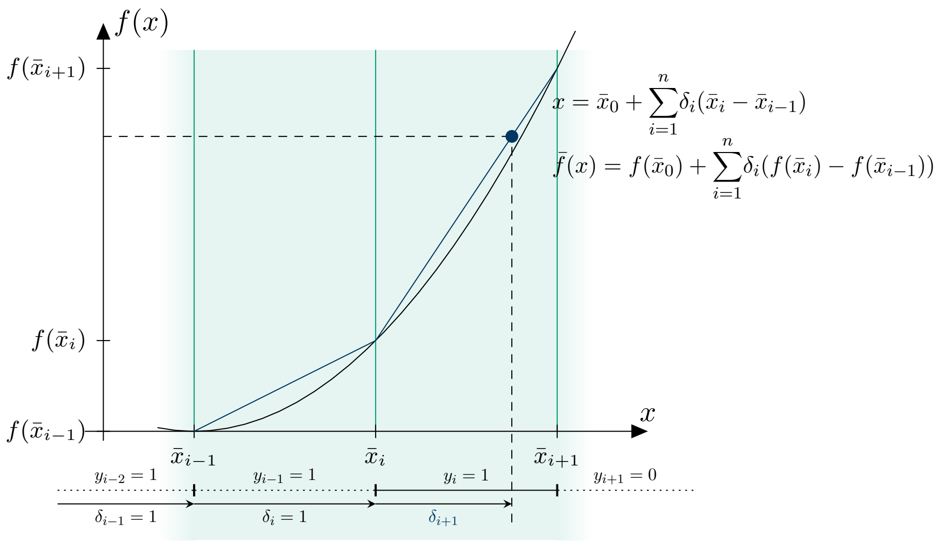


FIGURE 4. Representation of (Inc)

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

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

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 9. Classical Incremental Method

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

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. For each segment i, there is a variable x_i that represents the exact value on the x-axis. Every variable x_i yields one constraint of the form

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

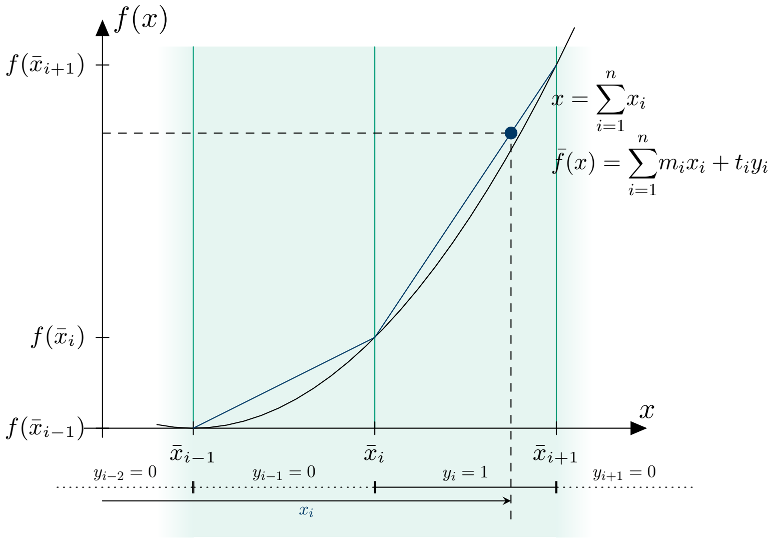


FIGURE 5. Visual representation of (MC)

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, one 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, one obtains (MC) and the visual representation in Figure 5.

Model 10. Multiple Choice Model

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

Finally, we provide an overview of the number of variables and constraints that are needed to model the MILP representations of this section in Table 2. For relaxations, we need another variable for each segment, i.e., n additional variables. This is the same for each representation.

TABLE 2. Sizes of all one-dimensional representations, assuming 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	$\lceil\log_2(n-1)\rceil$

4. Implementation

After introducing the different possibilities to represent MILP relaxations of MINLPs, we go more into detail about how we implemented our framework and how we conducted the computational study. First, we describe the input instances before heading over to the reformulation and solving process.

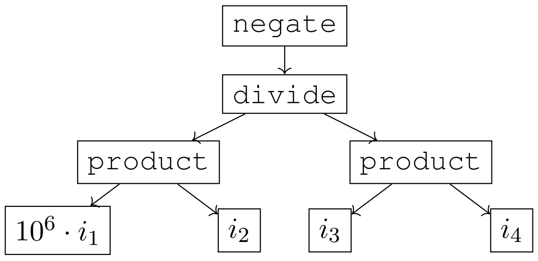
4.1. **Input.** All input problems have to be stored in the Optimization Services Instance Language, short OSIL, format. In case, that a problem is not available in this format, our framework provides a conversion tool for `.nl` files. The OSIL format employs an XML vocabulary, which provides several 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 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 [14].

Our benchmark set consists of 773 instances. We use a subset of the problems provided in the MINLPLIB ([9]) that can be represented by the

```
<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.

following nonlinear functions:

$$\begin{aligned} &f(x_1, x_2) = x_1 \cdot x_2,\ f(x_1, x_2) = \frac{x_1}{x_2},\ f(x) = \ln x,\ f(x) = \log_{10} x, \\ &f(x) = x^a \text{ (including } x^{-1} \text{ and } \sqrt{x}),\ f(x) = a^x \text{ (including } e^x), \\ &f(x) = \sin x,\ f(x) = \cos x,\ f(x) = \tanh x,\ f(x) = |x|. \end{aligned} \tag{1}$$

Furthermore, all variables must have both, lower and upper bounds; otherwise, a piecewise linear approximation or relaxation cannot be established. Using this selection criterion, only 306 instances remain in the benchmark set. To expand the set, we first apply presolving to the instances before selection. Presolving can establish missing bounds, allowing us to increase the benchmark set to 773 instances. Presolving is performed using SCIP [6]. 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 $z = x_1 x_2$. Subsequently, the univariate transformation of bivariate products from Section 2 is performed. The resulting MINLP then serves as the basis for the various MILP relaxations.

Our data format for storing MINLPs 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. Therefore, 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 $\texttt{product}(x_1, x_2, \dots, x_{n-1}, x_n)$. This is reformulated such that each product has two multiplicands, resulting in the form $\texttt{product}(x_1, \texttt{product}(x_2, \texttt{product}(\dots, \texttt{product}(x_{n-1}, x_n))))$. 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. Then, products of the form $x_1 x_2$ are reformulated using Equations (2) to (4).

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. Until now, the optimal solution for the resulting MINLP has not changed.

4.2.2. *Reformulation from one-dimensional MINLPs to MILP relaxations.* We now reformulate the MINLP based on the one-dimensional formulation using the MILP representations described in Section 3. Our implementation allows one to choose between relaxations and approximations; for the computational study conducted in this work we only consider piecewise linear relaxations. This relaxation has a larger set of feasible solutions than the initial MINLP; Its size is determined by the error bound used.

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

$$\left|\bar{f}(x) - f(x)\right| \leq \epsilon \tag{30}$$

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 approximation errors apply to each PWL function separately. When multiple approximations are used in the model, the cumulative effect can influence the final solution quality. In the worst case, the total error could scale with the number of PWL functions n, i.e., $\mathcal{O}(n\epsilon)$. However, optimization effects could reduce this. Later in Section 5.2.2, we will look into the solution quality, i.e, the practical impact of these errors.

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 (30) 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

$$\bar{f}(x) = m_i x + t_i, \tag{31}$$

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, one 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 MILP relaxation for each different representation that can be solved using state-of-the-art MILP solution methods. To accomplish this, we first convert the MILPs from our own data structure to Pyomo models, cf. [10, 18]. We use Pyomo because it allows us to easily switch between different solvers. In our study, we use Gurobi Optimizer version 11.0.3 ([17]) to solve the MILPs 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 them 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 MILP models for PWL relaxations from Section 3 perform in practice.

5.1. **Benchmark instances.**

As mentioned previously, our benchmark set is formed from a subset of the MINLPLIB and consists of 773 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 415 variables and 647 constraints on average per instance. The median numbers are 119 and 144, 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 68 binary/integer variables, as well as 147 nonlinear constraints per instance. The median number of binary/integer variables is 20, the median number of nonlinear constraints is 21.

Table 3 contains statistics about the reformulated 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 MILP models for PWL relaxations.**

We now present the computational results where we test all PWL MILP 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. Afterwards, we 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

$$\sqrt[n]{\prod_{i=1}^{n} (t_i + s)} - s. \tag{32}$$

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

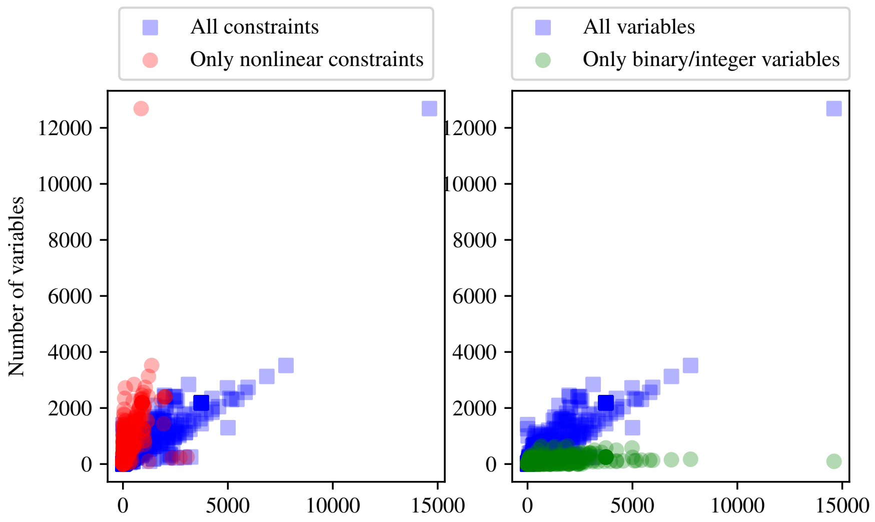
$$\sqrt[n]{\exp\left(\sum_{i=1}^{n} \ln (t_i + s)\right)} - s = \exp\left(\frac{\sum_{i=1}^{n} \ln (t_i + s)}{n}\right) - s. \tag{33}$$

Table 3. Sizes of the MILP reformulations. The instances have between 1 and 2818 nonlinearities, with a mean of 142.9 and a median of 24.

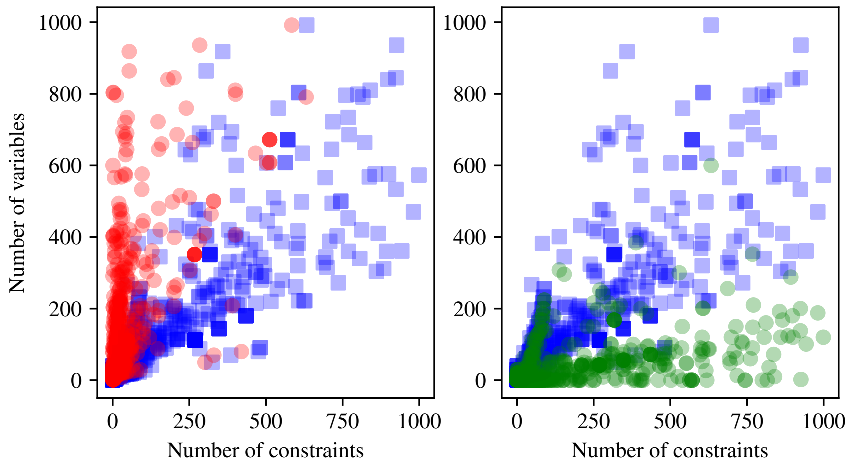
Error Bound	$\epsilon = 10^2$			
	Min	**Max**	**Mean**	**Median**
Segments per instance	2	752	46.78	28
Segments per NL	2	14	2.06	2
Segment length per NL	$1.36 \cdot 10^{-6}$	50000.00	13.53	1.50
Error Bound	$\epsilon = 10^0$			
	Min	**Max**	**Mean**	**Median**
Segments per instance	2	752	79.67	32
Segments per NL	2	126	3.52	2
Segment length per NL	$1.36 \cdot 10^{-6}$	9090.91	4.11	1.25
Error Bound	$\epsilon = 10^{-2}$			
	Min	**Max**	**Mean**	**Median**
Segments per instance	4	3856	480.02	123
Segments per NL	2	1249	21.19	7
Segment length per NL	$1.36 \cdot 10^{-6}$	1052.63	0.81	0.19
Error Bound	$\epsilon = 10^{-4}$			
	Min	**Max**	**Mean**	**Median**
Segments per instance	4	37386	4529.54	1027
Segments per NL	2	12474	199.96	53
Segment length per NL	$1.36 \cdot 10^{-6}$	117.51	0.12	0.02
Error Bound	$\epsilon = 10^{-6}$			
	Min	**Max**	**Mean**	**Median**
Segments per instance	4	373536	44959.64	10155
Segments per NL	2	124758	1984.82	521
Segment length per NL	$1.36 \cdot 10^{-6}$	13.01	0.02	$2.24 \cdot 10^{-3}$

In each MILP 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, our framework creates two variables $\epsilon^+, \epsilon^- \in [0, \epsilon]$ every time, a non-linearity is replaced.

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



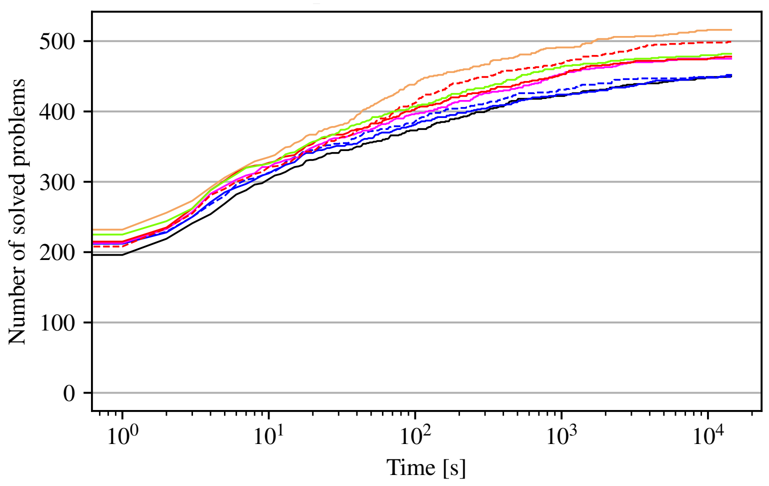
(7a) Full benchmark set



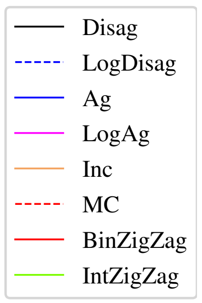
(7b) Instances with less than 1000 variables and constraints

Figure 7. Total amount of variables and constraints for our 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.

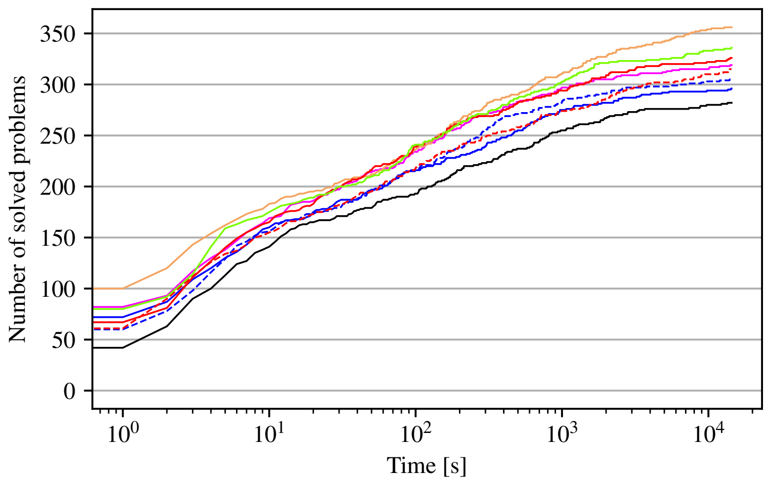
scale for better visibility. Each subfigure represents a different error bound (From now on, we will consider error bounds $\epsilon \in \{10^2, 10^0, 10^{-2}, 10^{-4}, 10^{-6}\}$ in all cases). All eight methods are plotted, and a legend is provided in Figure 8b. The bottom plots show more differences as the problems become larger and thus more difficult with increasing error bounds.



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



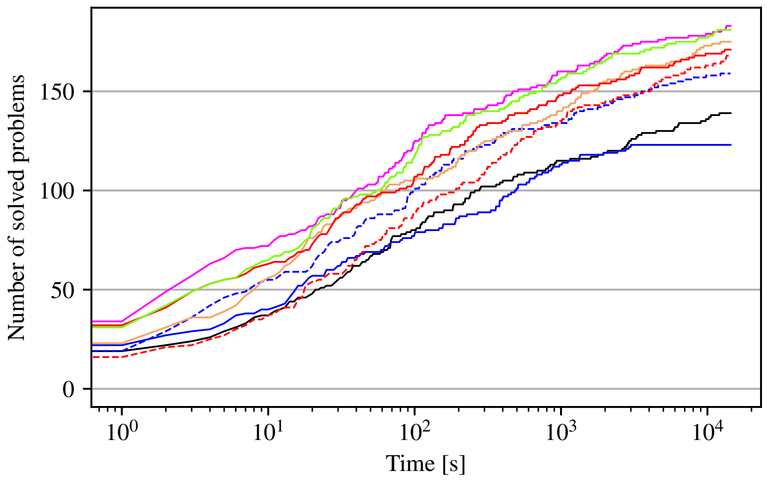
(8B) Legend



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

In Table 4, 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 a runtime. Additional to the mean and median values, we use the shifted geometric mean (SGM) with $s = 10$, as proposed in (32).

Table 5 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, 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 MILP model progressed how far in the solution process. For every model,



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

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

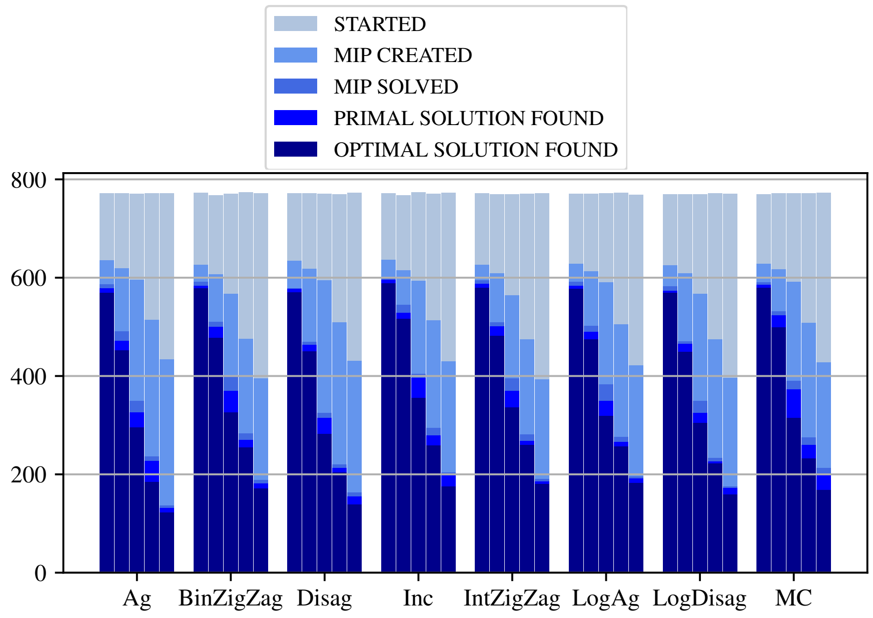


FIGURE 9. Solving progress of the different MILP methods. For each method, the bars represent the error bounds of $10^2, 10^0, 10^{-2}, 10^{-4}$, and 10^{-6}, from left to right.

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 MILP reformulation has been created (MIP CREATED), the MILP reformulation solution process has ended without errors (MIP SOLVED), and a primal

Table 4. 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)	847.67	0.13	16.56	2852.49	5.35	75.68
(LogDisag)	899.86	0.13	19.25	2757.08	4.49	65.41
(Ag)	898.24	0.11	17.57	2838.93	4.32	71.18
(LogAg)	664.25	0.10	15.24	2116.79	3.60	52.28
(Inc)	**394.55**	**0.08**	**8.76**	**1091.61**	**3.11**	**29.14**
(MC)	640.23	0.11	13.25	1541.92	3.82	40.91
(BinZigZag)	631.83	0.10	13.17	2085.42	3.42	49.18
(IntZigZag)	547.76	0.09	12.62	1966.44	3.50	45.23
Error bound	$\epsilon = 10^{-2}$			$\epsilon = 10^{-4}$		
Runtime [s]	**Mean**	**Median**	**SGM**	**Mean**	**Median**	**SGM**
(Disag)	5015.56	172.01	317.78	5062.35	207.74	371.20
(LogDisag)	4198.54	77.26	213.28	4201.27	68.10	231.36
(Ag)	4491.85	74.85	236.16	5785.87	200.44	429.07
(LogAg)	3724.75	**42.52**	163.06	2563.17	**29.91**	126.61
(Inc)	**2646.75**	44.21	**121.33**	2648.00	31.60	138.80
(MC)	4087.79	73.77	218.22	4055.82	60.13	238.43
(BinZigZag)	3551.82	43.28	158.89	2625.05	44.21	144.72
(IntZigZag)	3241.05	49.92	143.68	**2401.44**	37.79	**121.02**
Error bound	$\epsilon = 10^{-6}$					
Runtime [s]	**Mean**	**Median**	**SGM**			
(Disag)	6081.46	841.82	694.25			
(LogDisag)	4687.49	157.04	358.92			
(Ag)	6687.15	974.21	767.21			
(LogAg)	**3242.70**	**73.48**	**174.72**			
(Inc)	3898.44	191.25	286.14			
(MC)	4468.54	328.05	439.52			
(BinZigZag)	4000.26	126.88	258.30			
(IntZigZag)	3375.64	87.50	198.68			

or optimal solution of the relaxation has been found (PRIMAL/OPTIMAL SOLUTION FOUND), respectively.

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. 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)	570	29	450	100	282	137	204	98	139	86
(LogDisag)	569	30	449	101	305	114	222	80	159	66
(Ag)	569	30	452	98	296	123	185	117	123	102
(LogAg)	577	22	475	75	319	100	257	45	**183**	**42**
(Inc)	**589**	**10**	**516**	**34**	**356**	**63**	259	43	175	50
(MC)	579	20	499	51	315	104	232	70	168	57
(BinZigZag)	578	21	478	72	326	93	255	47	171	54
(IntZigZag)	580	19	482	68	336	83	**260**	**42**	181	44

5.2.2. *Solution qualities.* As we only solve relaxations, we further evaluate the gap between the optimal solution of the MINLP and the MILP relaxations' solution. Table 6 presents the corresponding results. Therein, additionally to the number of solved problems, we provide the median for the gaps between the optimal MINLP and MILP solutions x^* and x^{relax}, given by

$$\frac{\left|c^\top x^* - c^\top x^{\text{relax}}\right|}{|c^\top x^*| + 10^{-10}}. \tag{34}$$

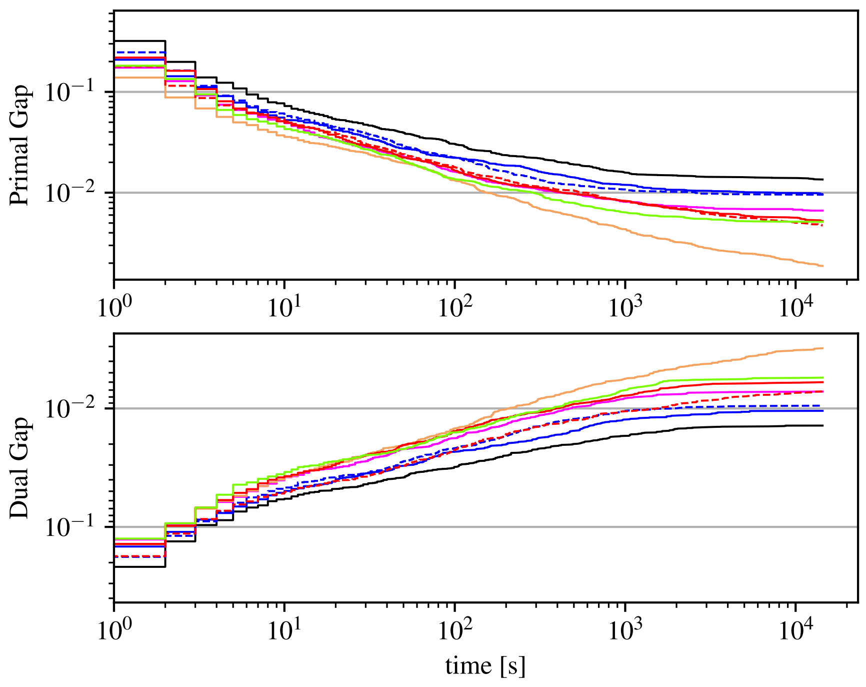
Adding 10^{-10} to the denominator prevents problems with instances that have an objective value of 0. As one can see, the gap is already very small for $\epsilon = 10^{-4}$, which means that using MILP relaxations, one can find fairly good dual bounds.

Table 6. Solution qualities of MILP relaxations, depending on the error bound ϵ, calculated by (34).

Error bound	$\epsilon = 10^2$	$\epsilon = 10^0$	$\epsilon = 10^{-2}$	$\epsilon = 10^{-4}$	$\epsilon = 10^{-6}$
Solved instances	556	428	267	156	103
Median gap	61.05%	30.56%	0.50%	0.01%	0.00%

Further, in Figure 10, we plot how the MILP 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 MILP solution. The primal gap is calculated using

$$p := \begin{cases} 1 & \text{if } \overline{z} = \infty \text{ or } \overline{z} \cdot z^* < 0, \\ 0 & \text{if } \overline{z} = z^*, \\ \frac{|\overline{z} - z^*|}{\max\{|\overline{z}|, |z^*|\}} & \text{else,} \end{cases} \tag{35}$$



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

and, equivalently, the dual gap is calculated by

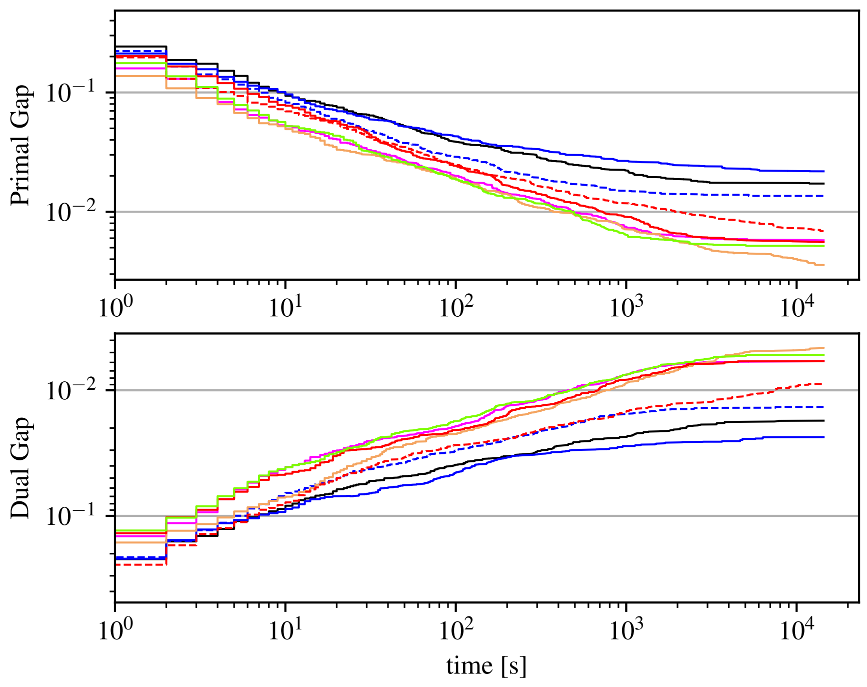
$$d := \begin{cases} 1 & \text{if } \underline{z} = -\infty \text{ or } \underline{z} \cdot z^* < 0, \\ 0 & \text{if } \underline{z} = z^*, \\ \frac{|\underline{z} - z^*|}{\max\{|\underline{z}|, |z^*|\}} & \text{else,} \end{cases} \tag{36}$$

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 MILP relaxation. The upper graphs in each subplot show how the primal gap shrinks, while the lower graphs show how the dual gap shrinks.

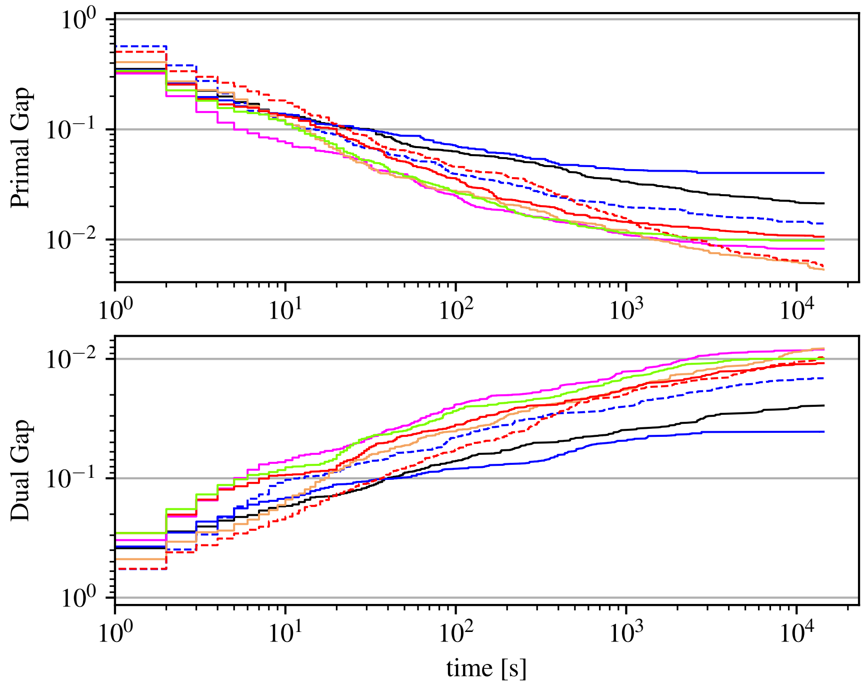
6. Discussion

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 for all methods, while already for $\epsilon = 1$ (Figure 8a), we can see some differences. For smaller error bounds, i.e., larger models, the 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 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 5, the non-logarithmic convex combinations, (Ag) and (Disag), solve the fewest instances. In general, the



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



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

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

logarithmic methods outperform their non-logarithmic counterparts, while the (IntZigZag) model is slightly superior to the other logarithmic models. The runtimes in Table 4 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 9 and 10), we see similar results as before: (Inc) solves the most instances, followed by the logarithmic models. For error bounds $\epsilon \geq 10^{-2}$, (Inc) again shows superior runtimes. This changes with $\epsilon = 10^{-4}$ and $\epsilon = 10^{-6}$, as now, (LogAg), (BinZigZag), and (IntZigZag) have similar or smaller runtimes, respectively. Again, (IntZigZag) performs best among the logarithmic methods. We can see the same results for the optimality subset, presented in Table 11, 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 12 to 14, 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 more 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 MILP 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 MILP and MINLP solutions, as described in (34). The optimal solution of a MILP relaxation is a valid dual bound for the corresponding MINLP. Thus, the results in Table 6, which shows the median values of all gaps, 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 MILP 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 MILP 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.

Remark 11. It is noteworthy that when evaluating the initial MINLPLIB problems instead of the presolved instances, one obtains similar results. This supports the reliability of our comparison and the conclusions made in this computational study.

7. Conclusion

In this paper, we compared various commonly used MILP models for PWL relaxations of nonlinear functions that are known in the literature. Using over 750 instances created from 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 [26] and are accompanied by a recommendation of this method for practical applications. Further, for very high-accuracy relaxations, the Zig-Zag formulations appear to be a viable alternative. Additionally, the multiple choice method offers a promising formulation for quickly finding feasible solutions. Practitioners can use our framework to conduct such a study for their type of problems since it is publicly available, see [7].

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 MILP 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 formulations.

References

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

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

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

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

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

[6] Suresh Bolusani, Mathieu Besançon, Ksenia Bestuzheva, Antonia Chmiela, João Dionísio, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Mohammed Ghannam, Ambros Gleixner, Christoph Graczyk, Katrin Halbig, Ivo Hedtke, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Dominik Kamp, Thorsten Koch, Kevin Kofler, Jurgen Lentz, Julian Manns, Gioni Mexi, Erik Mühmer, Marc E. Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Mark Turner, Stefan Vigerske, Dieter Weninger, and Lixing Xu. *The SCIP Optimization Suite 9.0*. ZIB-Report 24-02-29. Zuse Institute Berlin, Feb. 2024. URL: https://nbn-resolving.org/urn:nbn:de:0297-zib-95528.

[7] Kristin Braun. *GitHub - kristinbraun/pwl-t-rex — github.com*. https://github.com/kristinbraun/pwl-t-rex. 2025.

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

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

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

[11] Carlos M. Correa-Posada and Pedro Sánchez-Martín. “Gas Network Optimization: A comparison of Piecewise Linear Models”. 2014. URL: http://www.optimization-online.org/DB_HTML/2014/10/4580.html.

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

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

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

[15] Björn Geißler, Alexander Martin, Antonio Morsi, and Lars Schewe. “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, 2012, pp. 287–314. DOI: 10.1007/978-1-4614-1927-3_10.

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

[17] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2024. URL: https://www.gurobi.com.

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

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

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

[21] Joseph Andrew Huchette. “Advanced mixed-integer programming formulations : methodology, computation, and application”. 2018. URL: https://dspace.mit.edu/handle/1721.1/119282.

[22] Robert G Jeroslow and James K Lowe. *Modelling with integer variables*. Springer, 1984.

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

[24] Moritz Link and Stefan Volkwein. “Adaptive piecewise linear relaxations for enclosure computations for nonconvex multiobjective mixed-integer

quadratically constrained programs". In: *Journal of Global Optimization* 87.1 (2023), pp. 97–132. DOI: 10.1007/s10898-023-01309-5.

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

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

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

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

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

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

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

[32] Steffen Rebennack. "Computing tight bounds via piecewise linear functions through the example of circle cutting problems". In: *Mathematical Methods of Operations Research* 84 (1 2016), pp. 3–57. DOI: 10.1007/S00186-016-0546-0/FIGURES/15.

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

[34] Steffen Rebennack and Josef Kallrath. "Continuous piecewise linear delta-approximations for univariate functions: computing minimal breakpoint systems". In: *Journal of Optimization Theory and Applications* 167.2 (2015), pp. 617–643. DOI: 10.1007/s10957-014-0687-3.

[35] Steffen Rebennack and Vitaliy Krasko. "Piecewise Linear Function Fitting via Mixed-Integer Linear Programming". In: *INFORMS Journal on Computing* 32.2 (2019), pp. 507–530. DOI: 10.1287/ijoc.2019.0890.

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

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

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

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

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

Declarations

Funding. 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.

Competing Interests. There are no competing interests.

Author contributions. All authors contributed to the study’s conception and design. Robert Burlacu came up with the original idea. Kristin Braun was responsible for the implementation and the computational results. The proof of Lemma 5 was conducted by Kristin Braun. All authors contributed to the first draft of the manuscript, and all authors provided feedback on previous drafts. All authors read and approved the final manuscript.

Data availability. All data used during the current study is available in the MINLPLIB: https://www.minlplib.org/

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

The following Table 7 gives a visualization of our branching scheme from Section 3.2.

Appendix B. Benchmark set

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

Instance	Obj. Value	Constraints		Variables		
		Total	Nonlinear	Cont.	Binary	Integer
ann_peaks_exp	-6.563	98	47	100	0	0
arki0002	0.9768	1976	913	2456	0	0
arki0003	3795.2061	2583	1830	2283	0	0
arki0015	-272.2998	1496	1012	2093	0	0
arki0020	-41.075	2	1	1262	0	0
ball_mk2_10	0.0	1	1	0	0	10
ball_mk2_30	0.0	1	1	0	0	30

Instance	Obj. Value	Constraints		Variables		
		Total	Nonlinear	Cont.	Binary	Integer
ball_mk3_10	∞	1	1	0	0	10
ball_mk3_20	∞	1	1	0	0	20
ball_mk3_30	∞	1	1	0	0	30
ball_mk4_05	∞	1	1	0	0	10
ball_mk4_10	∞	1	1	0	0	20
ball_mk4_15	∞	1	1	0	0	30
bayes2_30	0.0001	77	55	86	0	0
bayes2_50	0.5202	77	55	86	0	0
beuster	116329.6706	114	47	105	52	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
btest14	-59.8174	93	86	135	0	0
camshape100	-4.2841	200	101	199	0	0
camshape200	-4.2785	400	201	399	0	0
camshape400	-4.2757	800	401	799	0	0
camshape800	-4.2743	1600	801	1599	0	0
carton7	191.7295	687	64	72	200	56
carton9	205.1371	893	68	72	216	72
cecil_13	-115656.4997	898	180	660	180	0
chakra	-179.1336	41	22	62	0	0
chance	29.8944	3	1	4	0	0
chenery	-1058.9199	38	23	43	0	0
chp_partload	23.2981	2516	490	2203	45	0
chp_shorttermplan1a	214.8424	2068	384	864	144	0
chp_shorttermplan2b	-162735.1963	2552	672	1200	192	0
clay0203hfsg	41573.2625	132	24	72	18	0
clay0203m	41573.2625	54	24	12	18	0
clay0204hfsg	6545.0	234	32	132	32	0
clay0204m	6545.0	90	32	20	32	0
clay0205hfsg	8092.5	365	40	210	50	0
clay0205m	8092.5	135	40	30	50	0
clay0303hfsg	26669.1096	150	36	78	21	0
clay0303m	26669.1096	66	36	12	21	0
clay0304hfsg	40262.3875	258	48	140	36	0
clay0304m	40262.3875	106	48	20	36	0
clay0305hfsg	8092.5	395	60	220	55	0
clay0305m	8092.5	155	60	30	55	0
crudeoil_lee1_05	79.75	1240	160	495	40	0
crudeoil_lee1_06	79.75	1503	192	594	48	0
crudeoil_lee1_07	79.75	1776	224	693	56	0
crudeoil_lee1_08	79.75	2059	256	792	64	0
crudeoil_lee1_09	79.75	2352	288	891	72	0
crudeoil_lee1_10	79.75	2655	320	990	80	0
crudeoil_lee2_05	96.1699	2581	420	1085	70	0
crudeoil_lee2_06	101.1746	3117	504	1302	84	0
crudeoil_lee2_07	101.1746	3670	588	1519	98	0
crudeoil_lee2_08	101.1746	4240	672	1736	112	0
crudeoil_lee2_10	101.1746	5431	840	2170	140	0
crudeoil_lee3_05	85.4489	2786	490	1210	70	0
crudeoil_lee3_06	85.4489	3359	588	1452	84	0
crudeoil_lee3_07	85.4489	3949	686	1694	98	0
crudeoil_lee3_08	85.4489	4556	784	1936	112	0
crudeoil_lee3_09	85.4489	5180	882	2178	126	0
crudeoil_lee3_10	85.4489	5821	980	2420	140	0
crudeoil_lee4_05	132.5476	4241	760	1860	95	0
crudeoil_lee4_06	132.5476	5093	912	2232	114	0
crudeoil_lee4_07	132.5476	5965	1064	2604	133	0
crudeoil_lee4_08	132.5476	6857	1216	2976	152	0
crudeoil_lee4_09	132.5476	7769	1368	3348	171	0
crudeoil_li01	5122.5645	695	56	296	48	0
crudeoil_li02	101567417.2	5004	54	1057	240	0
crudeoil_li03	3484.8292	2442	192	832	132	0
crudeoil_li05	3132.364	1916	192	808	132	0
crudeoil_li06	3355.0	2436	192	832	132	0
crudeoil_pooling_ct2	10246.22	732	70	295	108	0
crudeoil_pooling_ct4	13258.2597	924	95	395	138	0
csched1	-30639.2578	23	1	14	63	0
csched2	-166101.9964	138	1	93	308	0
cvxnonsep_normcon20	-21.7491	1	1	10	0	10
cvxnonsep_normcon20r	-21.7491	21	20	30	0	10
cvxnonsep_normcon30	-34.244	1	1	15	0	15
cvxnonsep_normcon30r	-34.244	31	30	45	0	15

Instance	Obj. Value	Constraints		Variables		
		Total	Nonlinear	Cont.	Binary	Integer
cvxnonsep_normcon40	-32.6297	1	1	20	0	20
cvxnonsep_normcon40r	-32.6297	41	40	60	0	20
cvxnonsep_nsig20	80.9493	1	1	10	0	10
cvxnonsep_nsig20r	80.9493	21	20	30	0	10
cvxnonsep_nsig30	130.6287	1	1	15	0	15
cvxnonsep_nsig30r	156.4267	31	30	45	0	15
cvxnonsep_nsig40	133.9613	1	1	20	0	20
cvxnonsep_nsig40r	133.9613	41	40	60	0	20
cvxnonsep_pcon20	-21.5123	1	1	10	0	10
cvxnonsep_pcon20r	-21.5123	20	19	29	0	10
cvxnonsep_pcon30	-35.9868	1	1	15	0	15
cvxnonsep_pcon30r	-35.9868	30	29	44	0	15
cvxnonsep_pcon40	-46.5992	1	1	20	0	20
cvxnonsep_pcon40r	-46.5992	40	39	59	0	20
cvxnonsep_psig20	93.8114	0	1	10	0	10
cvxnonsep_psig20r	95.8974	22	21	32	0	10
cvxnonsep_psig30	78.9989	0	1	15	0	15
cvxnonsep_psig40	85.4958	0	1	20	0	20
cvxnonsep_psig40r	86.5451	42	41	62	0	20
edgecross10-060	459.0	481	1	47	44	0
edgecross10-080	1037.0	481	1	17	74	0
edgecross14-078	725.0	1457	1	28	155	0
edgecross14-156	4310.0	1457	1	67	116	0
eg_int_s	6.4531	28	28	5	0	3
eniplac	-132117.083	189	24	117	24	0
ex1221	7.6672	5	2	2	3	0
ex1223a	4.5796	9	5	3	4	0
ex1223b	4.5796	9	5	3	4	0
ex1225	31.0	10	1	2	6	0
ex1226	-17.0	5	1	2	3	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
ex14_1_1	-0.0	4	4	3	0	0
ex14_1_4	-0.0	4	4	3	0	0
ex14_1_8	0.0	4	4	3	0	0
ex14_1_9	-0.0	2	2	2	0	0
ex14_2_2	0.0	5	4	4	0	0
ex14_2_5	0.0	5	4	4	0	0
ex14_2_8	0.0	5	4	4	0	0
ex14_2_9	0.0	5	4	4	0	0
ex2_1_1	-17.0	1	1	5	0	0
ex2_1_2	-213.0	2	1	6	0	0
ex2_1_4	-11.0	5	1	6	0	0
ex2_1_6	-39.0	5	1	10	0	0
ex2_1_9	-0.375	1	1	10	0	0
ex3_1_1	7049.248	6	3	8	0	0
ex3_1_2	-30665.5387	6	7	5	0	0
ex3_1_3	-310.0	6	3	6	0	0
ex3_1_4	-4.0	3	1	3	0	0
ex3pb	68.0097	31	5	24	8	0
ex4	-8.0641	30	26	11	25	0
ex4_1_1	-7.4873	0	1	1	0	0
ex4_1_2	-663.5001	0	1	1	0	0
ex4_1_3	-443.6717	0	1	1	0	0
ex4_1_4	0.0	0	1	1	0	0
ex4_1_6	7.0	0	1	1	0	0
ex4_1_7	-7.5	0	1	1	0	0
ex4_1_9	-5.508	2	2	2	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_3_2	1.8642	16	9	22	0	0
ex5_4_2	7512.2301	6	3	8	0	0
ex7_2_2	-0.3888	5	5	6	0	0
ex7_3_1	0.3417	7	1	4	0	0
ex7_3_6	∞	17	10	17	0	0
ex8_1_1	-2.0218	0	1	2	0	0
ex8_1_2	-1.0709	0	1	1	0	0
ex8_3_13	-43.0895	72	54	115	0	0
ex8_3_2	-0.4123	76	49	110	0	0

Instance	Obj. Value	Constraints		Variables		
		Total	Nonlinear	Cont.	Binary	Integer
ex8_3_3	-0.4166	76	49	110	0	0
ex8_3_4	-3.58	76	49	110	0	0
ex8_3_5	-0.0691	76	49	110	0	0
ex8_3_8	-3.2561	93	65	126	0	0
ex8_3_9	-0.763	45	27	78	0	0
ex9_1_4	-37.0	9	4	10	0	0
ex9_1_5	-1.0	12	5	13	0	0
ex9_1_8	-3.25	12	5	14	0	0
ex9_2_3	0.0	15	6	16	0	0
feedtray	-13.406	91	62	90	7	0
feedtray2	0.0	284	147	52	36	0
flay02h	37.9473	51	2	42	4	0
flay03h	48.9898	144	3	110	12	0
flay03m	48.9898	24	3	14	12	0
flay04h	54.4059	282	4	210	24	0
flay04m	54.4059	42	4	18	24	0
flay05h	64.4981	465	5	342	40	0
flay05m	64.4981	65	5	22	40	0
flay06h	66.9328	693	6	506	60	0
flay06m	66.9328	93	6	26	60	0
fo7	20.7298	211	14	72	42	0
fo7_2	17.7493	211	14	72	42	0
fo7_ar25_1	23.0936	269	14	70	0	42
fo7_ar2_1	24.8398	269	14	70	0	42
fo7_ar3_1	22.5175	269	14	70	0	42
fo7_ar4_1	20.7298	269	14	70	0	42
fo7_ar5_1	17.7493	269	14	70	0	42
fo8	22.3819	273	16	90	56	0
fo8_ar25_1	28.0452	347	16	88	0	56
fo8_ar2_1	30.3406	347	16	88	0	56
fo8_ar3_1	23.9101	347	16	88	0	56
fo8_ar4_1	22.3819	347	16	88	0	56
fo8_ar5_1	22.3819	347	16	88	0	56
fo9	23.4643	343	18	110	72	0
fo9_ar25_1	32.1864	435	18	108	0	72
fo9_ar2_1	32.625	435	18	108	0	72
fo9_ar3_1	24.8155	435	18	108	0	72
fo9_ar4_1	23.4643	435	18	108	0	72
fo9_ar5_1	23.4643	435	18	108	0	72
forest	14396226.49	309	24	163	73	0
gabriel01	45.2444	467	48	143	72	0
gabriel02	39.6097	597	96	190	71	0
gabriel04	9.2266	943	128	260	101	0
gabriel05	∞	1795	192	519	256	0
gams02	89466860.66	14608	865	12592	96	0
gasnet	6999381.562	69	44	80	10	0
gasprod_sarawak01	-32445.4049	212	34	93	38	0
gasprod_sarawak16	-32271.218	2252	544	1488	38	0
gastrans	89.0858	149	24	85	21	0
gastrans040	0.0	553	135	231	0	48
gastrans135	0.0	2472	510	961	0	232
gastrans582_cold13	0.0	3732	909	1936	0	250
gastrans582_cold13_95	0.0	3732	909	1936	0	250
gastrans582_cold17	0.0	3732	909	1936	0	250
gastrans582_cold17_95	0.0	3732	909	1936	0	250
gastrans582_cool12	0.0	3732	909	1936	0	250
gastrans582_cool12_95	0.0	3732	909	1936	0	250
gastrans582_cool14	0.0	3732	909	1936	0	250
gastrans582_cool14_95	0.0	3732	909	1936	0	250
gastrans582_freezing27	∞	3732	909	1936	0	250
gastrans582_freezing27_95	∞	3732	909	1936	0	250
gastrans582_freezing30	0.0	3732	909	1936	0	250
gastrans582_freezing30_95	0.0	3732	909	1936	0	250
gastrans582_mild10	0.0	3732	909	1936	0	250
gastrans582_mild10_95	0.0	3732	909	1936	0	250
gastrans582_mild11	0.0	3732	909	1936	0	250
gastrans582_mild11_95	0.0	3732	909	1936	0	250
gastrans582_warm15	0.0	3732	909	1936	0	250
gastrans582_warm31	0.0	3732	909	1936	0	250
gastrans582_warm31_95	0.0	3732	909	1936	0	250
gear	0.0	0	1	0	0	4
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
genpooling_meyer10	1086187.137	423	33	207	187	0

Instance	Obj. Value	Constraints		Variables		
		Total	Nonlinear	Cont.	Binary	Integer
genpooling_meyer15	943734.0436	768	48	382	352	0
ghg_2veh	7.7709	62	48	39	18	0
ghg_3veh	7.754	119	88	60	36	0
gkocis	-1.9231	8	2	8	3	0
gsg_0001	2378.1605	112	1	78	0	0
haverly	-400.0	9	3	12	0	0
hda	-5964.5341	718	148	709	13	0
heatexch_gen1	154895.933	120	48	100	12	0
heatexch_trigen	1001973.081	262	1	246	45	0
himmel16	-0.866	21	21	18	0	0
house	-4500.0	8	3	8	0	0
hydroenergy1	209721.0066	428	48	192	96	0
hydroenergy2	371811.847	856	96	384	192	0
hydroenergy3	744963.7246	1498	168	672	336	0
inscribedsquare01	0.9901	8	9	8	0	0
inscribedsquare02	0.968	8	9	8	0	0
kall_circles_c6a	2.1117	54	22	18	0	0
kall_circles_c6b	1.9736	54	22	18	0	0
kall_circles_c6c	2.7977	63	29	20	0	0
kall_circles_c7a	2.6628	69	29	20	0	0
kall_circles_c8a	2.5409	86	37	22	0	0
kall_circlespolygons_c1p11	0.1996	48	21	43	0	0
kall_circlespolygons_c1p12	0.3396	48	21	43	0	0
kall_circlespolygons_c1p13	0.3396	48	21	43	0	0
kall_circlespolygons_c1p5a	2.8487	174	106	158	0	0
kall_circlespolygons_c1p5b	3.7696	816	631	791	0	0
kall_circlespolygons_c1p6a	3.744	1134	904	1110	0	0
kall_circlesrectangles_c1r11	0.1996	52	23	49	0	0
kall_circlesrectangles_c1r12	0.3396	52	23	49	0	0
kall_circlesrectangles_c1r13	0.2146	52	23	49	0	0
kall_circlesrectangles_c6r1	7.1645	192	133	184	0	0
kall_circlesrectangles_c6r29	6.2952	388	283	390	0	0
kall_circlesrectangles_c6r39	6.1752	619	466	634	0	0
kall_congruentcircles_c31	0.6438	16	4	10	0	0
kall_congruentcircles_c32	1.3759	16	4	10	0	0
kall_congruentcircles_c41	0.8584	24	7	12	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_c61	1.2876	46	16	16	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_10	11.9355	71	46	24	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
kall_diffcircles_8	14.4813	49	29	20	0	0
kall_diffcircles_9	13.3503	60	37	22	0	0
kall_ellipsoids_tc02b	32.4	128	48	124	0	0
kall_ellipsoids_tc03c	36.4536	196	74	193	0	0
kall_ellipsoids_tc05a	39.3979	461	321	464	0	0
kport40	37.1758	48	38	153	3	111
kriging_peaks-full010	0.2911	24	20	26	0	0
kriging_peaks-full020	0.3724	44	40	46	0	0
kriging_peaks-full030	-1.5866	64	60	66	0	0
kriging_peaks-full050	-1.1566	104	100	106	0	0
kriging_peaks-full100	-2.6375	204	200	206	0	0
kriging_peaks-full200	-3.8902	404	400	406	0	0
kriging_peaks-full500	-4.928	1004	1000	1006	0	0
kriging_peaks-red010	0.2911	0	1	2	0	0
kriging_peaks-red020	0.3724	0	1	2	0	0
kriging_peaks-red030	-1.5866	0	1	2	0	0
kriging_peaks-red050	-1.1566	0	1	2	0	0
kriging_peaks-red100	-2.6375	0	1	2	0	0
kriging_peaks-red200	-3.8902	0	1	2	0	0
kriging_peaks-red500	-4.928	0	1	2	0	0
m3	37.8	43	6	20	6	0
m6	82.2569	157	12	56	30	0
m7	106.7569	211	14	72	42	0
m7_ar25_1	143.585	269	14	70	0	42
m7_ar2_1	190.235	269	14	70	0	42
m7_ar3_1	143.585	269	14	70	0	42
m7_ar4_1	106.7569	269	14	70	0	42

Instance	Obj. Value	Constraints		Variables		
		Total	Nonlinear	Cont.	Binary	Integer
m7_ar5_1	106.46	269	14	70	0	42
mathopt4	0.0	2	2	2	0	0
mathopt5_1	-0.9996	0	1	1	0	0
mathopt5_2	-1.0	0	1	1	0	0
mathopt5_3	-1.6164	0	1	1	0	0
mathopt5_4	0.0	0	1	1	0	0
mathopt5_5	-14.838	0	1	1	0	0
mathopt5_7	-4.4367	0	1	1	0	0
mathopt5_8	-0.6861	0	1	1	0	0
mathopt6	-3.3069	0	1	2	0	0
maxmin	-0.3661	78	78	27	0	0
maxmineig2	0.3955	111	90	200	0	0
milinfract	2.6339	501	1	500	500	0
multiplants_stg1	355.0866	262	34	217	198	0
multiplants_stg1a	390.9655	250	25	208	216	0
multiplants_stg1b	471.7496	280	28	233	243	0
multiplants_stg1c	708.4403	270	22	225	252	0
multiplants_stg5	5843.273	299	25	235	216	0
multiplants_stg6	5166.1213	388	33	310	384	0
ndcc12	106.3542	237	46	598	46	0
ndcc12persp	106.3542	283	46	644	46	0
ndcc13	84.625	254	42	588	42	0
ndcc13persp	85.8919	296	42	630	42	0
ndcc14	110.3276	305	54	810	54	0
ndcc14persp	111.2697	359	54	864	54	0
ndcc15	94.6112	306	40	640	40	0
ndcc15persp	94.6112	346	40	680	40	0
ndcc16	112.0715	377	60	1020	60	0
ndcc16persp	113.5459	437	60	1080	60	0
no7_ar25_1	107.8153	269	14	70	0	42
no7_ar2_1	107.8153	269	14	70	0	42
no7_ar3_1	107.8153	269	14	70	0	42
no7_ar4_1	98.5184	269	14	70	0	42
no7_ar5_1	90.6227	269	14	70	0	42
nuclear14	-1.1297	1226	602	986	576	0
nuclear14a	-1.1295	633	584	392	600	0
nuclear14b	-1.1276	1785	560	968	600	0
nuclear25	-1.1199	1303	628	1053	625	0
nuclear25a	-1.1207	659	608	408	650	0
nuclear25b	-1.1144	1909	583	1033	650	0
nuclearva	-1.0142	317	267	183	168	0
nuclearvb	-1.0313	317	267	183	168	0
nuclearvc	-1.0042	317	267	183	168	0
nuclearvd	-1.0405	317	267	183	168	0
nuclearve	-1.0376	317	267	183	168	0
nuclearvf	-1.0241	317	267	183	168	0
nvs04	0.72	0	1	0	0	2
nvs06	1.7703	0	1	0	0	2
nvs09	-43.1343	0	1	0	0	10
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
nvs15	1.0	1	1	0	0	3
nvs16	0.7031	0	1	0	0	2
nvs17	-1100.4	7	8	0	0	7
nvs18	-778.4	6	7	0	0	6
nvs21	-5.6848	2	3	1	0	2
nvs23	-1125.2	9	10	0	0	9
nvs24	-1033.2	10	11	0	0	10
o7	131.6531	211	14	72	42	0
o7_2	116.9459	211	14	72	42	0
o7_ar25_1	140.412	269	14	70	0	42
o7_ar2_1	140.412	269	14	70	0	42
o7_ar3_1	137.9318	269	14	70	0	42
o7_ar4_1	131.6531	269	14	70	0	42
o7_ar5_1	116.9458	269	14	70	0	42
o8_ar4_1	243.0707	347	16	88	0	56
o9_ar4_1	236.1385	435	18	108	0	72
oaer	-1.9231	7	2	6	3	0
oil	-0.9325	1546	418	1516	19	0
oil2	-0.7333	926	284	934	2	0
ortez	-9532.0391	74	27	69	18	0
orth_d3m6	0.7071	62	52	25	0	0
orth_d3m6_pl	0.7071	127	66	42	0	0
orth_d4m6_pl	0.6495	86	41	42	0	0

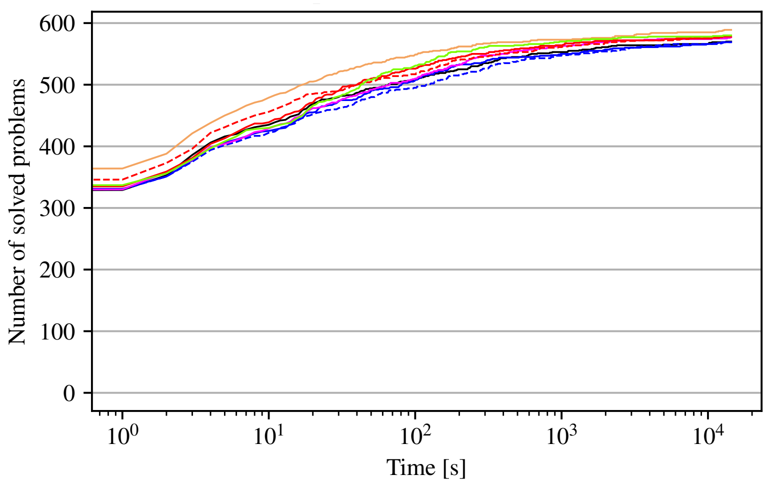
Instance	Obj. Value	Constraints		Variables		
		Total	Nonlinear	Cont.	Binary	Integer
p_ball_10b_5p_2d_h	18.7186	219	50	130	50	0
p_ball_10b_5p_2d_m	18.7186	109	50	30	50	0
p_ball_10b_5p_3d_h	44.0042	294	50	195	50	0
p_ball_10b_5p_4d_h	71.3719	369	50	260	50	0
p_ball_10b_5p_4d_m	71.3719	149	50	60	50	0
p_ball_10b_7p_3d_h	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_10p_2d_h	41.934	1149	300	710	300	0
p_ball_30b_10p_2d_m	41.934	529	300	110	300	0
p_ball_30b_5p_2d_h	0.2916	539	150	330	150	0
p_ball_30b_5p_2d_m	0.2916	229	150	30	150	0
p_ball_30b_5p_3d_h	8.2183	714	150	495	150	0
p_ball_30b_5p_3d_m	8.2183	249	150	45	150	0
p_ball_30b_7p_2d_h	13.9338	771	210	476	210	0
p_ball_30b_7p_2d_m	13.9338	337	210	56	210	0
p_ball_40b_5p_3d_h	9.7772	924	200	645	200	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
pindyck	-1170.4863	96	32	116	0	0
pointpack02	2.0	3	1	5	0	0
pointpack04	1.0	10	6	9	0	0
pointpack06	0.3611	21	15	13	0	0
pointpack08	0.2679	36	28	17	0	0
pointpack10	0.1775	55	45	21	0	0
pointpack12	0.1511	78	66	25	0	0
pointpack14	0.1217	105	91	29	0	0
polygon25	-0.7797	324	301	50	0	0
polygon50	-0.7839	1274	1226	100	0	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_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_digabel16	-2410.6877	117	81	171	0	0
pooling_digabel18	-689.1606	412	390	208	0	0
pooling_digabel19	-4539.9122	171	128	212	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

Instance	Obj. Value	Constraints		Variables		
		Total	Nonlinear	Cont.	Binary	Integer
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_sppa0pq	-35812.3336	744	329	500	0	0
pooling_sppa0stp	-35812.3336	1083	658	616	0	0
pooling_sppa0tp	-35812.3336	744	329	500	0	0
pooling_sppa5pq	-27915.8392	1383	968	1245	0	0
pooling_sppa5stp	-27829.022	2361	1936	1441	0	0
pooling_sppa5tp	-27922.0748	1383	968	1245	0	0
pooling_sppa9pq	-21933.994	2407	1992	2399	0	0
pooling_sppa9tp	-21933.994	2407	1992	2399	0	0
pooling_sppb0pq	-43412.4119	1957	1153	1537	0	0
pooling_sppb0tp	-43372.8152	1957	1153	1537	0	0
portfol_classical050_1	-0.0948	103	1	100	50	0
portfol_classical200_2	-0.1101	403	1	400	200	0
portfol_robust050_34	-0.0721	156	2	152	51	0
portfol_robust100_09	-0.105	306	2	302	101	0
portfol_robust200_03	-0.1291	606	2	602	201	0
portfol_shortfall050_68	-1.0972	157	2	153	51	0
portfol_shortfall100_04	-1.1179	307	2	303	101	0
portfol_shortfall200_05	-1.1273	607	2	603	201	0
prob03	10.0	1	1	0	0	2
prob06	1.1771	2	2	2	0	0
prob09	-0.0	1	1	3	0	0
prob10	3.4455	2	1	1	0	1
procsel	-1.9231	7	2	7	3	0
procurement2mot	212.0707	761	12	736	60	0
product	-2142.9481	1925	132	1446	107	0
product2	-2102.3893	3125	528	2714	128	0
rbrock	0.0	0	1	2	0	0
ringpack_10_1	-20.0665	385	330	20	50	0
ringpack_10_2	-20.0665	475	420	20	60	0
ringpack_20_1	-35.1696	2547	2337	40	175	0
ringpack_20_2	-39.3413	2927	2717	40	195	0
ringpack_20_3	-38.6317	3228	3055	40	213	0
rocket100	-1.0128	502	502	607	0	0
rocket200	-1.0128	1002	1002	1207	0	0
rocket400	-1.0128	2002	2002	2407	0	0
rocket50	-1.0128	252	252	307	0	0
routingdelay_bigm	146.6256	2977	396	727	396	0
routingdelay_proj	146.6256	2977	396	727	396	0
rsyn0805hfsg	1296.1206	429	3	271	37	0
rsyn0805m	1296.1206	286	3	101	69	0
rsyn0805m02hfsg	2238.3954	1045	6	552	148	0
rsyn0805m02m	2238.3954	769	6	212	148	0
rsyn0805m03hfsg	3068.9314	1698	9	828	222	0
rsyn0805m03m	3068.9314	1284	9	318	222	0
rsyn0805m04hfsg	7174.219	2438	12	1104	296	0
rsyn0805m04m	7174.219	1886	12	424	296	0
rsyn0810hfsg	1721.4477	483	6	301	42	0
rsyn0810m	1721.4477	312	6	111	74	0
rsyn0810m02hfsg	1741.3869	1188	12	622	168	0
rsyn0810m02m	1741.3869	866	12	242	168	0
rsyn0810m03hfsg	2722.448	1935	18	933	252	0
rsyn0810m03m	2722.448	1452	18	363	252	0
rsyn0810m04hfsg	6581.9344	2784	24	1244	336	0
rsyn0810m04m	6581.9344	2140	24	484	336	0
rsyn0815hfsg	1269.9256	552	11	340	47	0
rsyn0815m	1269.9256	347	11	126	79	0
rsyn0815m02hfsg	1774.3973	1361	22	710	188	0
rsyn0815m02m	1774.3973	981	22	282	188	0
rsyn0815m03hfsg	2827.9259	2217	33	1065	282	0
rsyn0815m03m	2827.9259	1647	33	423	282	0
rsyn0815m04hfsg	3410.8543	3190	44	1420	376	0
rsyn0815m04m	3410.8543	2430	44	564	376	0
rsyn0820hfsg	1150.3005	604	14	365	52	0
rsyn0820m	1150.3005	371	14	131	84	0
rsyn0820m02hfsg	1092.0911	1500	28	770	208	0
rsyn0820m02m	1092.0911	1074	28	302	208	0
rsyn0820m03m	2028.8119	1809	42	453	312	0
rsyn0820m04hfsg	2450.7722	3528	56	1540	416	0
rsyn0820m04m	2450.7722	2676	56	604	416	0
rsyn0830hfsg	510.072	716	20	432	62	0
rsyn0830m	510.072	425	20	156	94	0
rsyn0830m02hfsg	730.5072	1794	40	924	248	0
rsyn0830m02m	730.5072	1272	40	372	248	0

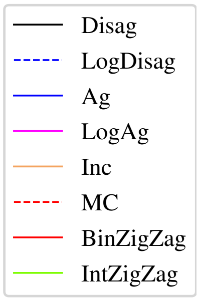
Instance	Obj. Value	Constraints		Variables		
		Total	Nonlinear	Cont.	Binary	Integer
rsyn0830m03hfsg	1543.0593	2934	60	1386	372	0
rsyn0830m03m	1543.0593	2151	60	558	372	0
rsyn0830m04hfsg	2529.0734	4236	80	1848	496	0
rsyn0830m04m	2529.0734	3192	80	744	496	0
rsyn0840hfsg	325.5545	837	28	496	72	0
rsyn0840m	325.5545	484	28	176	104	0
rsyn0840m02hfsg	734.9835	2106	56	1072	288	0
rsyn0840m02m	734.9835	1480	56	432	288	0
rsyn0840m03m	2742.6457	2508	84	648	432	0
rsyn0840m04hfsg	2564.4995	4980	112	2144	576	0
rsyn0840m04m	2564.4995	3728	112	864	576	0
sep1	-510.081	31	6	27	2	0
sfacloc1_2_80	12.7521	2088	15	169	62	0
sfacloc1_2_90	17.8916	348	15	169	30	0
sfacloc1_2_95	18.8501	208	15	162	9	0
sfacloc1_3_80	8.5231	2161	15	231	62	0
sfacloc1_3_90	11.622	421	15	231	30	0
sfacloc1_3_95	12.3025	281	15	224	9	0
sfacloc1_4_80	7.8791	2234	15	293	62	0
sfacloc1_4_90	10.4575	494	15	293	30	0
sfacloc1_4_95	11.1841	354	15	286	9	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
spectra2	13.9783	72	8	39	30	0
squfl010-025persp	214.111	525	250	500	10	0
squfl010-040persp	240.5985	840	400	800	10	0
squfl010-080persp	509.706	1680	800	1600	10	0
squfl015-060persp	366.6218	1860	900	1800	15	0
squfl015-080persp	402.4885	2480	1200	2400	15	0
squfl020-040persp	209.2549	1640	800	1600	20	0
squfl020-050persp	230.2021	2050	1000	2000	20	0
squfl025-025persp	168.8072	1275	625	1250	25	0
squfl025-030persp	205.5017	1530	750	1500	25	0
squfl025-040persp	197.3339	2040	1000	2000	25	0
sssd08-04	182022.5703	40	12	16	44	0
sssd08-04persp	182022.5703	40	12	16	44	0
sssd12-05	281408.6352	52	15	20	75	0
sssd12-05persp	281408.6353	52	15	20	75	0
sssd15-04	205054.4585	47	12	16	72	0
sssd15-04persp	205054.4585	47	12	16	72	0
sssd15-06	539635.4697	63	18	24	108	0
sssd15-06persp	539635.4697	63	18	24	108	0
sssd15-08	562617.8818	79	24	32	144	0
sssd15-08persp	562617.8818	79	24	32	144	0
sssd16-07	417188.8105	72	21	28	133	0
sssd16-07persp	417188.8105	72	21	28	133	0
sssd18-06	397992.2951	66	18	24	126	0
sssd18-06persp	397992.2951	66	18	24	126	0
sssd18-08	832795.5852	82	24	32	168	0
sssd18-08persp	832795.5852	82	24	32	168	0
sssd20-04persp	347691.4105	52	12	16	92	0
sssd20-08	469619.8376	84	24	32	184	0
sssd20-08persp	469619.8376	84	24	32	184	0
sssd22-08	508713.7312	86	24	32	200	0
sssd22-08persp	508713.731	86	24	32	200	0
sssd25-04	300176.5637	57	12	16	112	0
sssd25-04persp	300176.5637	57	12	16	112	0
sssd25-08	472093.078	89	24	32	224	0
sssd25-08persp	472093.078	89	24	32	224	0
st_bpaf1a	-45.3797	10	1	10	0	0
st_bpaf1b	-42.9626	10	1	10	0	0
st_bpk1	-13.0	6	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_e05	7049.2493	3	2	5	0	0
st_e06	0.0	3	1	3	0	0
st_e07	-400.0	7	3	10	0	0

Instance	Obj. Value	Constraints		Variables		
		Total	Nonlinear	Cont.	Binary	Integer
st_e08	0.7418	2	2	2	0	0
st_e09	-0.5	1	2	2	0	0
st_e12	-4.5142	3	1	4	0	0
st_e13	2.0	2	1	1	1	0
st_e15	7.6672	5	2	2	3	0
st_e18	-2.8284	4	2	2	0	0
st_e19	-118.7049	2	2	2	0	0
st_e21	-13.4019	6	1	6	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_e30	-1.5811	15	5	14	0	0
st_e31	-2.0	135	5	88	24	0
st_e33	-400.0	6	3	9	0	0
st_e34	0.0156	4	4	6	0	0
st_e37	0.001	1	1	4	0	0
st_e40	30.4142	8	4	1	0	3
st_e41	641.8236	2	3	4	0	0
st_glmp_fp3	-12.0	8	1	4	0	0
st_glmp_kk92	-12.0	8	1	4	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_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_pan1	-5.2837	4	1	3	0	0
st_ph1	-230.1173	5	1	6	0	0
st_ph11	-11.2812	4	1	3	0	0
st_ph12	-22.625	4	1	3	0	0
st_ph13	-11.2812	10	1	3	0	0
st_ph14	-229.7222	10	1	3	0	0
st_ph15	-392.7037	4	1	4	0	0
st_ph2	-1028.1173	5	1	6	0	0
st_ph3	-420.2348	5	1	6	0	0
st_qpc-m3a	-382.695	10	1	10	0	0
st_qpc-m3b	0.0	10	1	10	0	0
st_qpk2	-12.25	12	1	6	0	0
st_qpk3	-36.0	22	1	11	0	0
st_robot	0.0	8	7	8	0	0
st_rv1	-59.9439	5	1	10	0	0
st_rv2	-64.4807	10	1	20	0	0
st_rv3	-35.7607	20	1	20	0	0
st_rv7	-138.1875	20	1	30	0	0
st_rv8	-132.6616	20	1	40	0	0
st_rv9	-120.1531	20	1	50	0	0
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
syn05hfsg	837.7324	58	3	37	5	0
syn05m02hfsg	3032.7354	151	6	84	20	0
syn05m02m	3032.7354	101	6	40	20	0
syn05m03hfsg	4027.3718	249	9	126	30	0
syn05m03m	4027.3718	174	9	60	30	0
syn05m04hfsg	5510.3873	362	12	168	40	0
syn05m04m	5510.3873	262	12	80	40	0
syn10hfsg	1267.3536	112	6	67	10	0
syn10m	1267.3536	54	6	25	10	0
syn10m02hfsg	2310.3007	294	12	154	40	0
syn10m02m	2310.3007	198	12	70	40	0
syn10m03hfsg	3354.6828	486	18	231	60	0
syn10m03m	3354.6828	342	18	105	60	0
syn10m04hfsg	4557.0623	708	24	308	80	0
syn10m04m	4557.0623	516	24	140	80	0
syn15hfsg	853.2847	181	11	106	15	0
syn15m	853.2847	89	11	40	15	0
syn15m02hfsg	2832.7489	467	22	242	60	0
syn15m02m	2832.7489	313	22	110	60	0
syn15m03hfsg	3850.1818	768	33	363	90	0
syn15m03m	3850.1818	537	33	165	90	0
syn15m04hfsg	4937.4777	1114	44	484	120	0
syn15m04m	4937.4777	806	44	220	120	0
syn20hfsg	924.2633	233	14	131	20	0

Instance	**Obj. Value**	**Constraints**		**Variables**		
		Total	**Nonlinear**	**Cont.**	**Binary**	**Integer**
syn20m	924.2633	113	14	45	20	0
syn20m02hfsg	1752.1332	606	28	302	80	0
syn20m02m	1752.1332	406	28	130	80	0
syn20m03hfsg	2646.9509	999	42	453	120	0
syn20m03m	2646.9509	699	42	195	120	0
syn20m04hfsg	3532.7439	1452	56	604	160	0
syn20m04m	3532.7439	1052	56	260	160	0
syn30hfsg	138.1596	345	20	198	30	0
syn30m	138.1596	167	20	70	30	0
syn30m02hfsg	399.6831	900	40	456	120	0
syn30m02m	399.6831	604	40	200	120	0
syn30m03hfsg	654.1542	1485	60	684	180	0
syn30m03m	654.1542	1041	60	300	180	0
syn30m04hfsg	865.722	2160	80	912	240	0
syn30m04m	865.722	1568	80	400	240	0
syn40hfsg	67.7133	466	28	262	40	0
syn40m	67.7133	226	28	90	40	0
syn40m02hfsg	388.7724	1212	56	604	160	0
syn40m02m	388.7724	812	56	260	160	0
syn40m03hfsg	395.148	1998	84	906	240	0
syn40m03m	395.148	1398	84	390	240	0
syn40m04hfsg	901.7511	2904	112	1208	320	0
syn40m04m	901.7511	2104	112	520	320	0
t1000	0.0	28	24	1002	0	0
tanksize	1.2686	74	21	38	9	0
tln12	90.5	72	12	0	12	156
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
tls2	5.3	24	2	4	31	2
tls4	8.3	64	4	16	85	4
tls5	10.3	90	5	25	131	5
tls6	15.3	120	6	36	173	6
tls7	15.0	154	7	49	289	7
tltr	48.0667	54	3	0	12	36
torsion25	-0.4175	4	2	1408	0	0
trig	-3.7625	1	2	1	0	0
util	999.5788	167	4	117	28	0
wastepaper3	0.0189	30	16	25	27	0
wastepaper4	0.0035	38	20	32	44	0
wastepaper5	0.0008	46	24	39	65	0
wastepaper6	0.0001	54	28	46	90	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
wastewater11m2	2127.1154	251	112	303	0	0
wastewater12m1	1201.0385	57	11	196	0	0
wastewater12m2	1201.0385	407	220	516	0	0
wastewater13m1	1564.958	83	16	382	0	0
wastewater13m2	1564.958	782	480	1039	0	0
wastewater14m1	513.0009	46	12	74	0	0
wastewater14m2	513.0009	204	90	208	0	0
wastewater15m1	2446.4286	40	12	46	0	0
wastewater15m2	2446.4286	151	48	133	0	0
waterno2_01	19.4567	204	65	157	9	0
waterno2_02	39.5714	410	130	314	18	0
waterno2_03	115.0045	616	195	471	27	0
waterno2_04	145.4398	822	260	628	36	0
waterno2_06	285.2266	1234	390	942	54	0
waterno2_09	933.2934	1852	585	1413	81	0
waterno2_12	2302.511	2470	780	1884	108	0
watertreatnd_conc	348336.7614	319	29	355	5	0
watertreatnd_flow	348337.0417	379	155	415	5	0
waterund01	86.8333	38	14	40	0	0
waterund08	164.4898	95	40	90	0	0
waterund11	104.8861	64	28	64	0	0
waterund14	329.5698	135	66	125	0	0
waterund17	157.0944	66	27	74	0	0
waterund18	238.7333	64	28	60	0	0



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



(11B) Legend

Instance	Obj. Value	Constraints		Variables		
		Total	Nonlinear	Cont.	Binary	Integer
`waterund22`	323.5051	135	66	146	0	0
`waterund25`	410.6354	87	36	121	0	0
`waterund27`	556.6752	208	96	432	0	0
`waterund28`	1812.1703	540	240	760	0	0
`waterund32`	638.7168	380	160	660	0	0
`waterund36`	662.807	239	110	324	0	0

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.

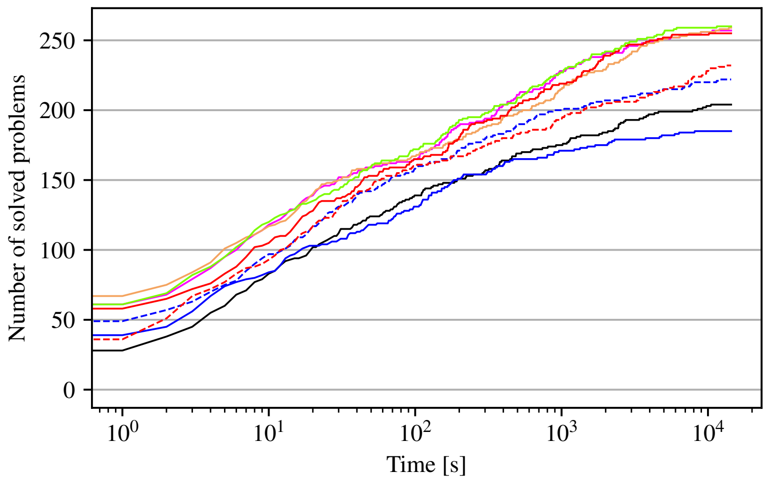
Further, we consider instances where any method finds at least one feasible solution, as this creates a subset of easier instances for each error bound. The solving results for this subset can be seen in Table 9. Further, in Table 10, we again provide the runtimes as in Table 4. Finally, we want to consider the subset of instances that were solved to optimality by all methods. The runtimes for this subset are given in Table 11.

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 12 to 14 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 limited to the time limit. Table 12 considers the full instance that, while Table 13 and Table 14 are limited to our easier subsets.

TABLE 7. 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

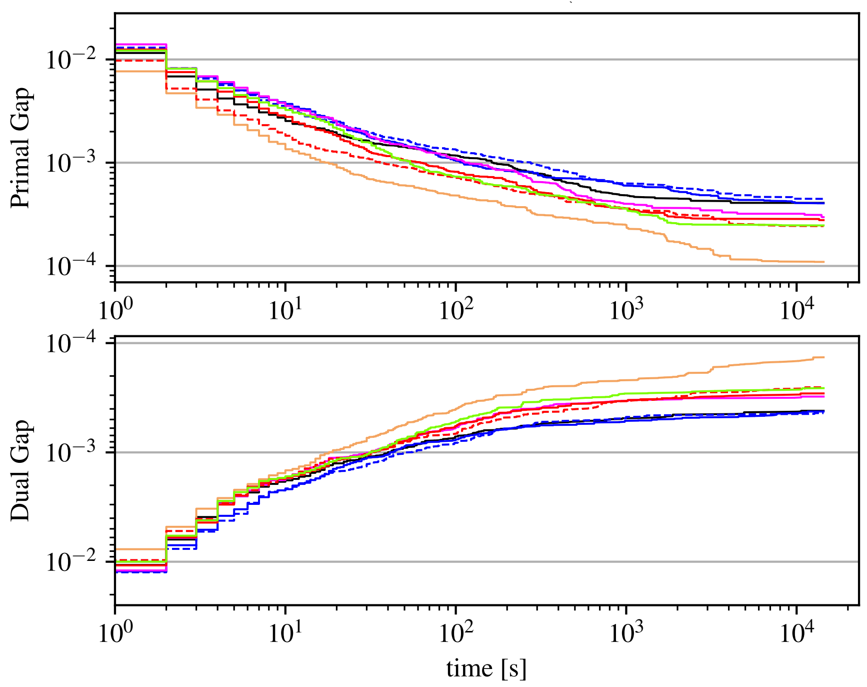


(11C) $\epsilon = 10^{-4}$

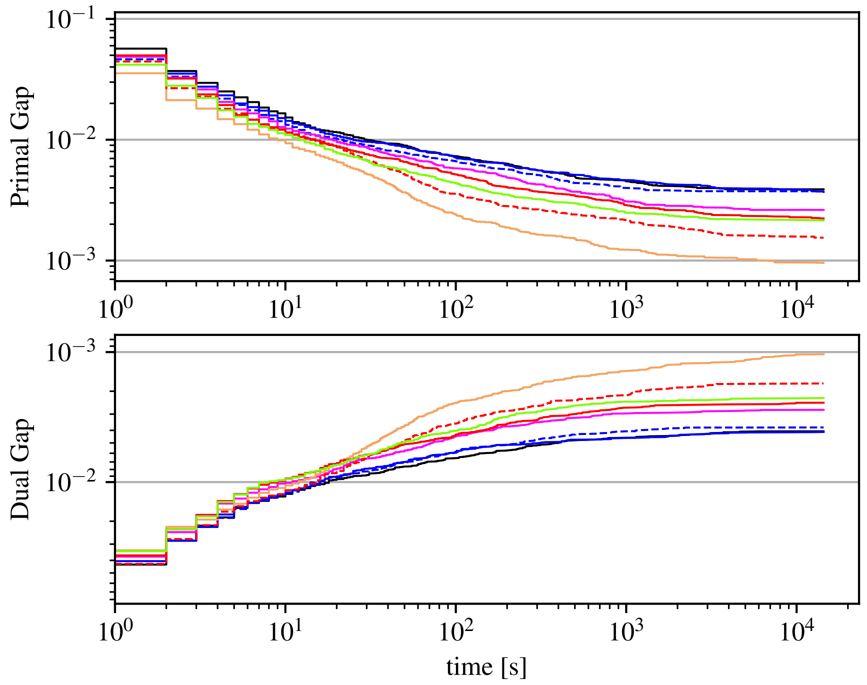
FIGURE 11. Number of solved problems over time for all considered error bounds.

TABLE 9. 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)	559	6	435	9	271	22	189	4	111	1
(LogDisag)	561	4	434	10	281	12	**192**	**1**	**112**	**0**
(Ag)	558	7	431	13	277	16	159	34	104	8
(LogAg)	561	4	**440**	**4**	283	10	**192**	**1**	**112**	**0**
(Inc)	**562**	**3**	**440**	**4**	**288**	**5**	191	2	**112**	**0**
(MC)	561	4	436	8	281	12	191	2	111	1
(BinZigZag)	561	4	439	5	283	10	**192**	**1**	**112**	**0**
(IntZigZag)	561	4	**440**	**4**	285	8	**192**	**1**	**112**	**0**



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



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

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

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 a feasible solution.

Error bound	$\epsilon = 10^{2}$			$\epsilon = 10^{0}$		
Runtime [s]	**Mean**	**Median**	**SGM**	**Mean**	**Median**	**SGM**
(Disag)	305.90	0.09	10.17	558.72	2.36	19.43
(LogDisag)	272.83	0.07	11.28	443.31	1.92	15.21
(Ag)	364.62	0.08	11.18	673.65	1.98	18.40
(LogAg)	216.73	0.08	9.39	242.83	1.81	12.69
(Inc)	**164.45**	**0.05**	**5.54**	**217.48**	**0.55**	**10.45**
(MC)	198.76	0.07	7.91	406.81	1.81	15.40
(BinZigZag)	186.82	0.08	7.85	287.86	1.73	11.94
(IntZigZag)	175.56	0.07	7.87	266.78	1.48	11.25
Error bound	$\epsilon = 10^{-2}$			$\epsilon = 10^{-4}$		
Runtime [s]	**Mean**	**Median**	**SGM**	**Mean**	**Median**	**SGM**
(Disag)	1438.34	11.79	65.36	828.23	18.54	53.19
(LogDisag)	871.64	7.46	40.60	443.39	11.80	31.60
(Ag)	1075.47	7.79	43.86	2698.21	16.93	90.99
(LogAg)	629.80	6.10	28.12	315.67	5.96	22.33
(Inc)	**404.97**	**3.55**	**24.73**	425.95	5.73	25.25
(MC)	1000.08	8.15	44.30	696.66	11.66	41.38
(BinZigZag)	674.39	6.18	29.05	303.16	7.79	24.91
(IntZigZag)	568.09	4.42	27.67	**263.14**	**5.72**	**20.99**
Error bound	$\epsilon = 10^{-6}$					
Runtime [s]	**Mean**	**Median**	**SGM**			
(Disag)	549.78	33.99	51.91			
(LogDisag)	81.34	12.09	22.17			
(Ag)	1236.75	24.21	67.16			
(LogAg)	30.02	**3.32**	**10.81**			
(Inc)	156.97	12.42	28.97			
(MC)	536.78	32.69	53.00			
(BinZigZag)	40.47	6.27	14.47			
(IntZigZag)	**29.81**	6.58	12.84			

C.4. **Performance profiles.** Additionally, we provide performance profile plots as proposed by [13] 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

Table 11. 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.

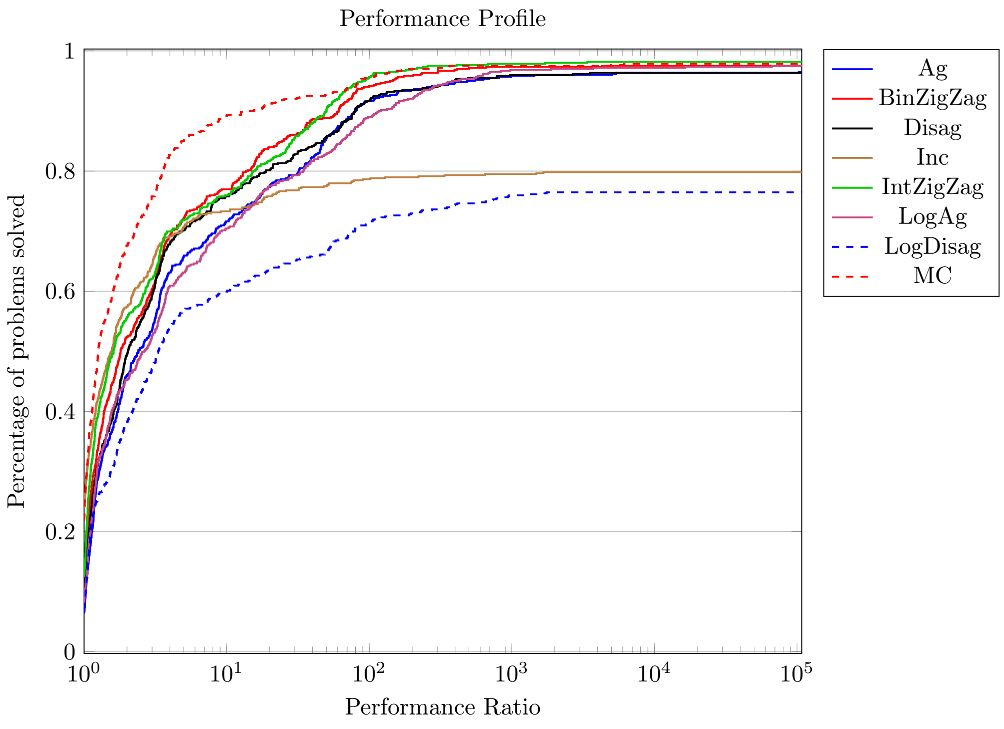
Error bound	$\epsilon = 10^2$			$\epsilon = 10^0$		
Runtime [s]	**Mean**	**Median**	**SGM**	**Mean**	**Median**	**SGM**
(Disag)	135.886	0.083	8.385	208.331	1.950	14.110
(LogDisag)	171.941	0.066	9.922	91.636	1.659	10.569
(Ag)	189.070	0.075	9.460	217.744	1.656	12.681
(LogAg)	115.377	0.073	8.209	60.467	1.591	9.136
(Inc)	**67.501**	**0.047**	**4.588**	**40.201**	**0.351**	**7.205**
(MC)	89.904	0.067	6.539	96.534	1.635	10.830
(BinZigZag)	84.676	0.076	6.720	54.319	1.534	8.295
(IntZigZag)	73.841	0.062	6.766	53.744	0.463	7.899
Error bound	$\epsilon = 10^{-2}$			$\epsilon = 10^{-4}$		
Runtime [s]	**Mean**	**Median**	**SGM**	**Mean**	**Median**	**SGM**
(Disag)	328.702	8.261	36.621	357.557	8.263	25.210
(LogDisag)	131.747	5.632	24.085	41.593	6.389	11.661
(Ag)	147.861	6.173	23.920	184.269	8.522	22.995
(LogAg)	67.517	4.697	16.769	**17.699**	3.169	6.954
(Inc)	**65.474**	**2.554**	**14.581**	17.905	**2.684**	7.417
(MC)	151.582	5.691	24.339	198.821	5.639	16.051
(BinZigZag)	72.542	4.361	17.150	18.037	4.329	8.480
(IntZigZag)	68.777	3.809	16.295	20.860	2.831	**6.788**
Error bound	$\epsilon = 10^{-6}$					
Runtime [s]	**Mean**	**Median**	**SGM**			
(Disag)	379.089	28.378	39.457			
(LogDisag)	32.892	8.343	16.006			
(Ag)	222.819	17.953	40.631			
(LogAg)	**14.026**	**2.578**	**7.716**			
(Inc)	121.671	9.109	22.196			
(MC)	348.243	19.168	39.168			
(BinZigZag)	20.329	3.926	10.448			
(IntZigZag)	17.843	4.206	9.788			

given factor of the best overall runtime. The performance profiles work as follows: Let $t_{p,s}$ be the runtime needed by MILP relaxation $\bar{s}$ to solve instance p. With the performance ratio $r_{p,\bar{s}} \coloneqq t_{p,\bar{s}} / \min_s t_{p,s}$, the performance profile function value $P(\tau)$ is the percentage of problems solved by approach

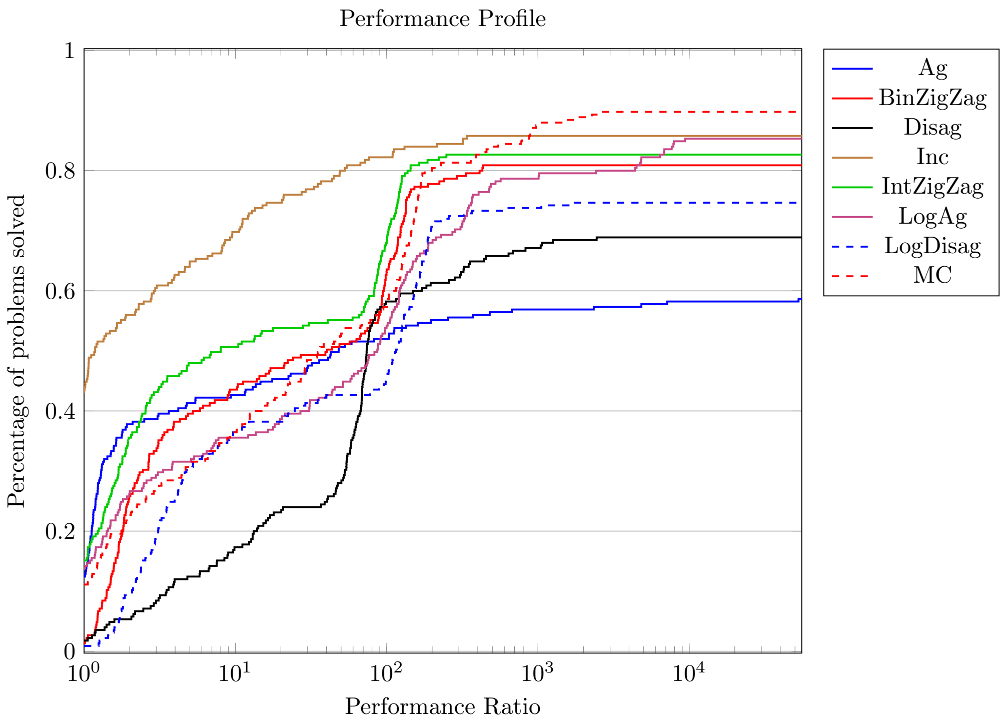
TABLE 12. Shifted geometric mean of the runtimes when the time limit is smaller.

Error bound	$\epsilon = 10^2$			$\epsilon = 10^0$		
Time limit	100s	1000s	10000s	100s	1000s	10000s
(Disag)	7.86	12.65	16.08	17.53	40.01	70.15
(LogDisag)	8.78	14.60	18.69	16.03	35.22	60.53
(Ag)	8.37	13.21	17.05	16.46	37.37	65.95
(LogAg)	8.15	12.32	14.87	15.29	31.98	49.26
(Inc)	**5.26**	**7.32**	**8.63**	**12.87**	**21.72**	**28.27**
(MC)	6.82	10.55	12.96	15.14	27.90	39.19
(BinZigZag)	7.21	10.59	12.85	14.70	30.15	46.38
(IntZigZag)	7.51	10.44	12.34	14.26	28.30	42.76
Error bound	$\epsilon = 10^{-2}$			$\epsilon = 10^{-4}$		
Time limit	100s	1000s	10000s	100s	1000s	10000s
(Disag)	36.30	118.89	280.62	41.04	136.06	328.65
(LogDisag)	32.76	92.40	191.96	34.55	99.03	208.93
(Ag)	32.26	97.01	210.84	41.01	139.41	371.27
(LogAg)	29.20	76.51	148.44	28.76	73.66	119.37
(Inc)	**27.18**	**68.52**	**114.25**	**28.47**	78.00	**130.99**
(MC)	32.48	94.64	197.82	34.48	102.78	217.96
(BinZigZag)	29.39	76.75	145.38	31.54	83.15	136.19
(IntZigZag)	28.69	73.80	132.75	28.84	**72.35**	114.49
Error bound	$\epsilon = 10^{-6}$					
Time limit	100s	1000s	10000s			
(Disag)	54.42	210.54	601.99			
(LogDisag)	44.60	140.61	321.23			
(Ag)	52.09	217.11	648.88			
(LogAg)	**34.47**	**89.41**	**161.90**			
(Inc)	40.71	128.61	262.78			
(MC)	52.71	179.96	397.81			
(BinZigZag)	39.17	114.88	235.30			
(IntZigZag)	37.09	99.16	184.02			

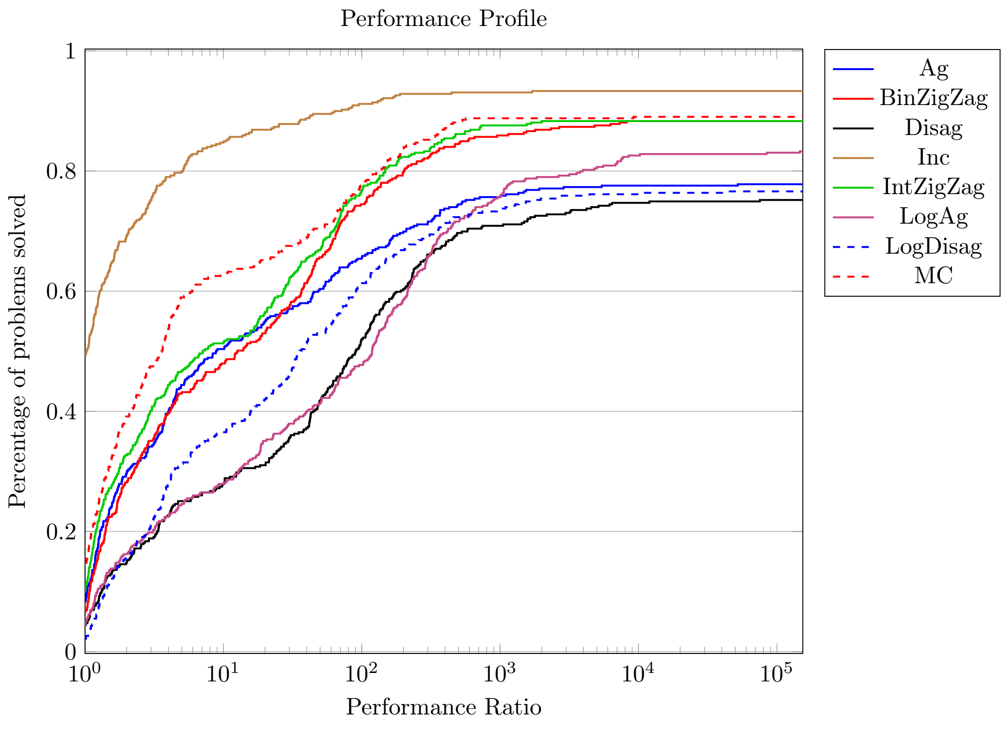
$\bar{s}$ such that the ratios $r_{p,\bar{s}}$ are within a factor $\tau \in \mathbb{R}$ of the best possible ratios. Figure 13 considers the time until a first feasible solution was found while Figure 14 considers the overall runtime until an instance was solved to optimality. All performance profiles are generated with the help of *perprof-py* ([38]).



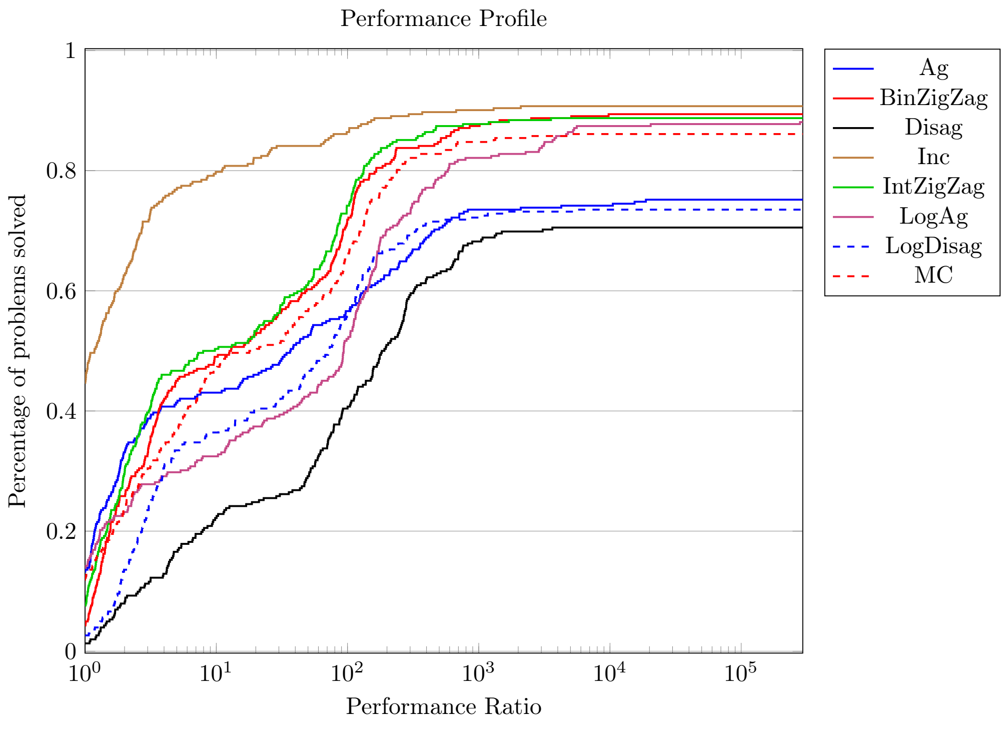
(13A) $\epsilon = 10^2$



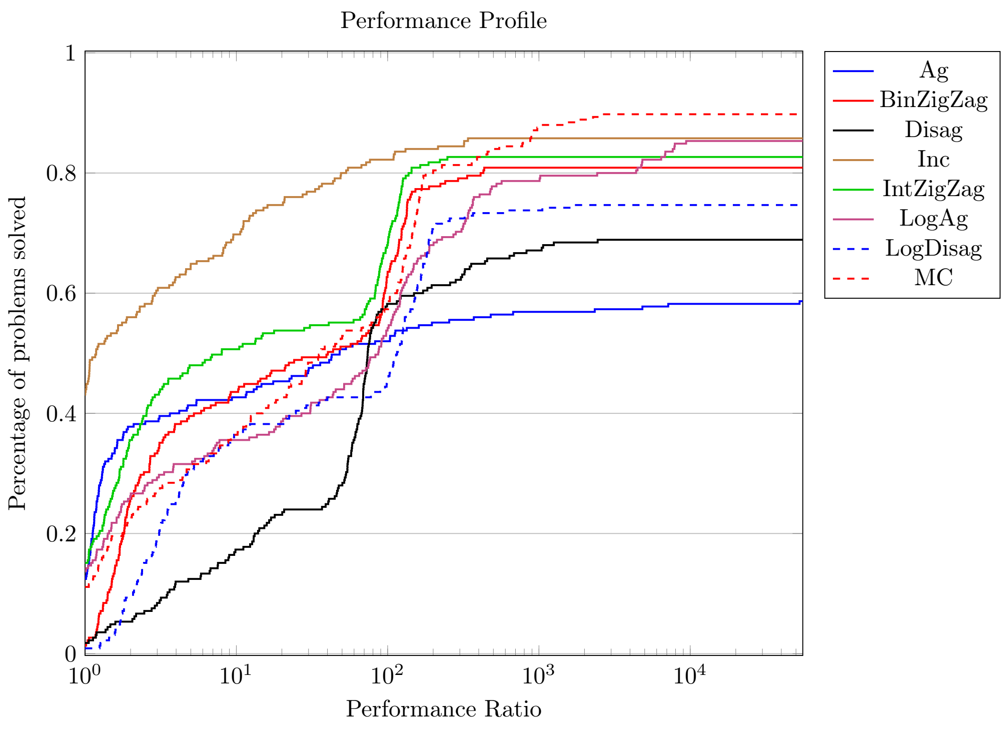
(13B) $\epsilon = 10^0$



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

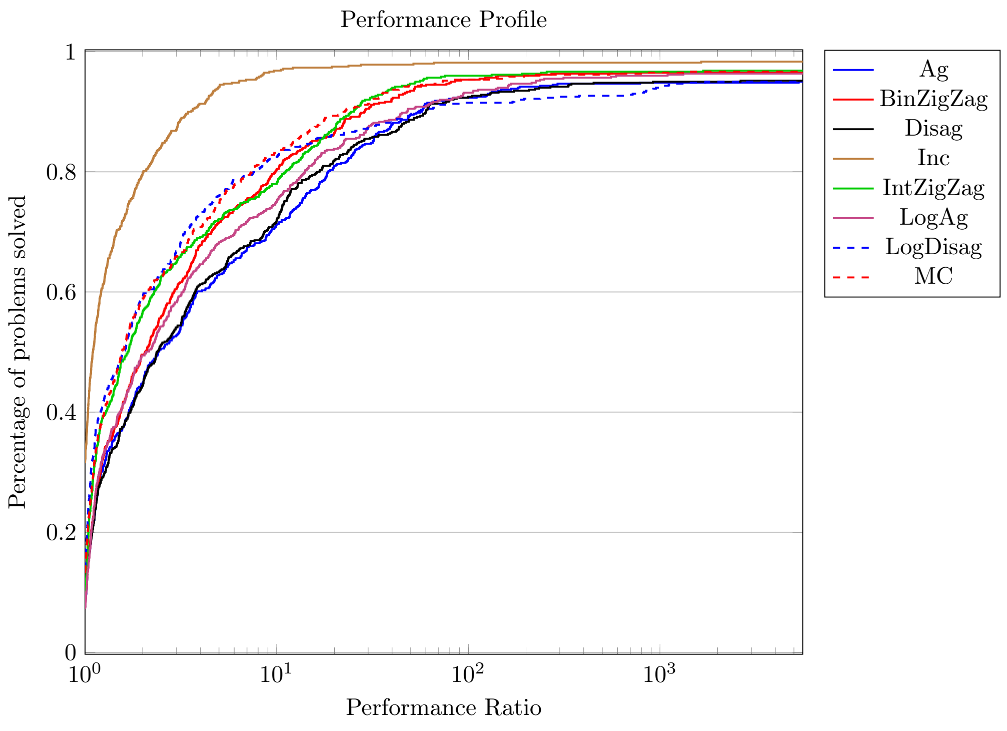


(13d) $\epsilon = 10^{-4}$

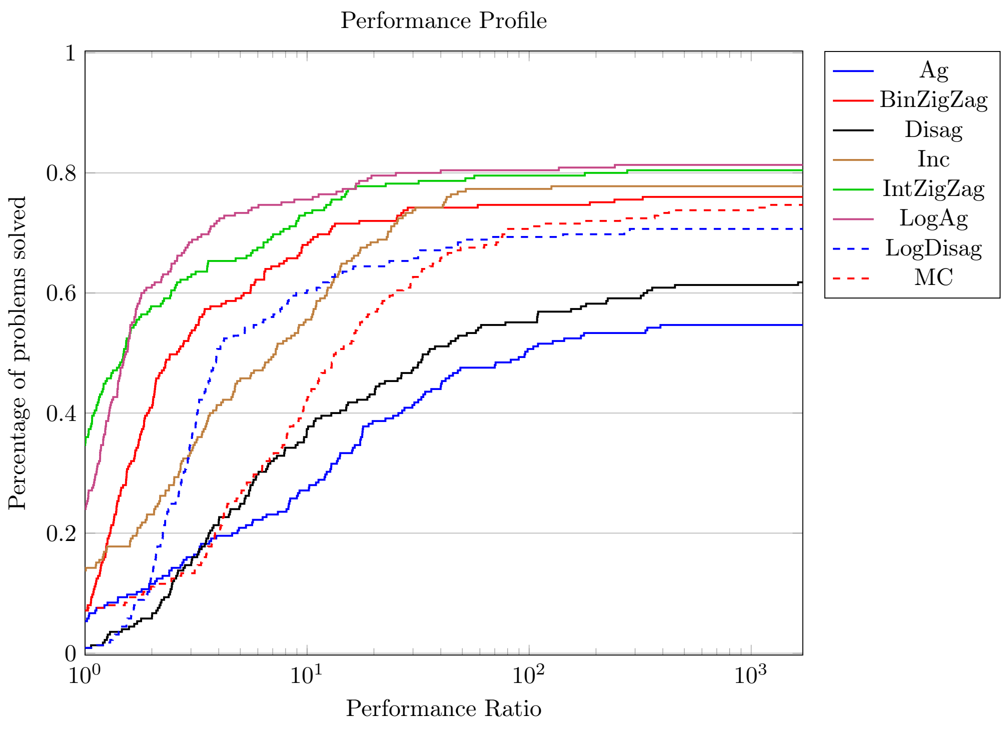


(13E) $\epsilon = 10^{-6}$

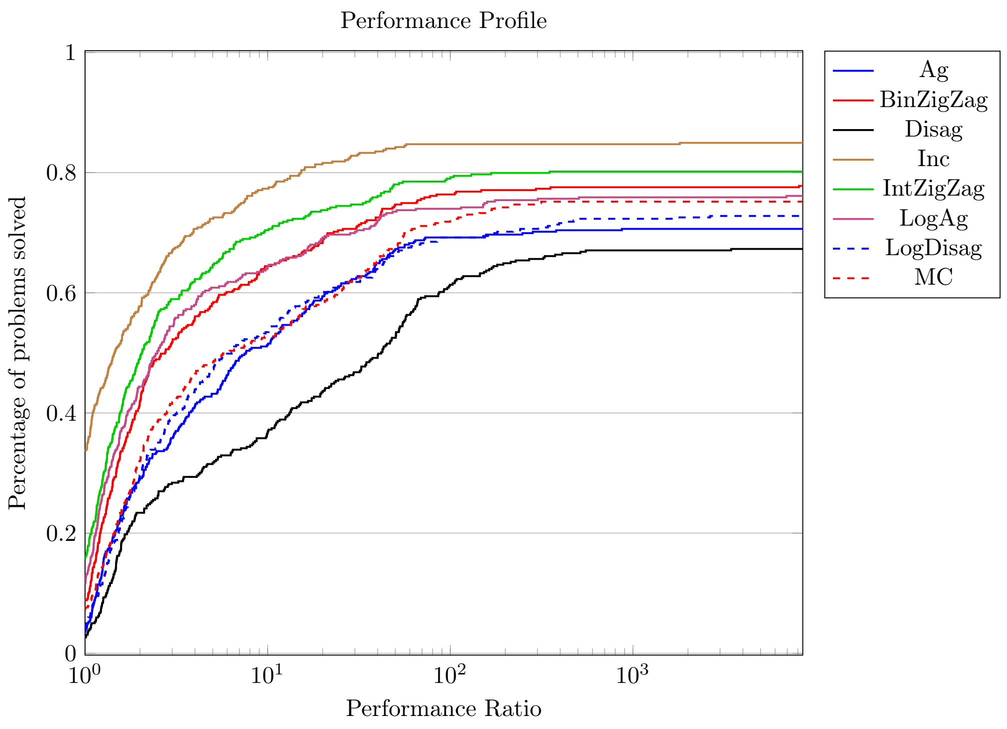
FIGURE 13. Performance profiles considering the time until a primal solution is found.



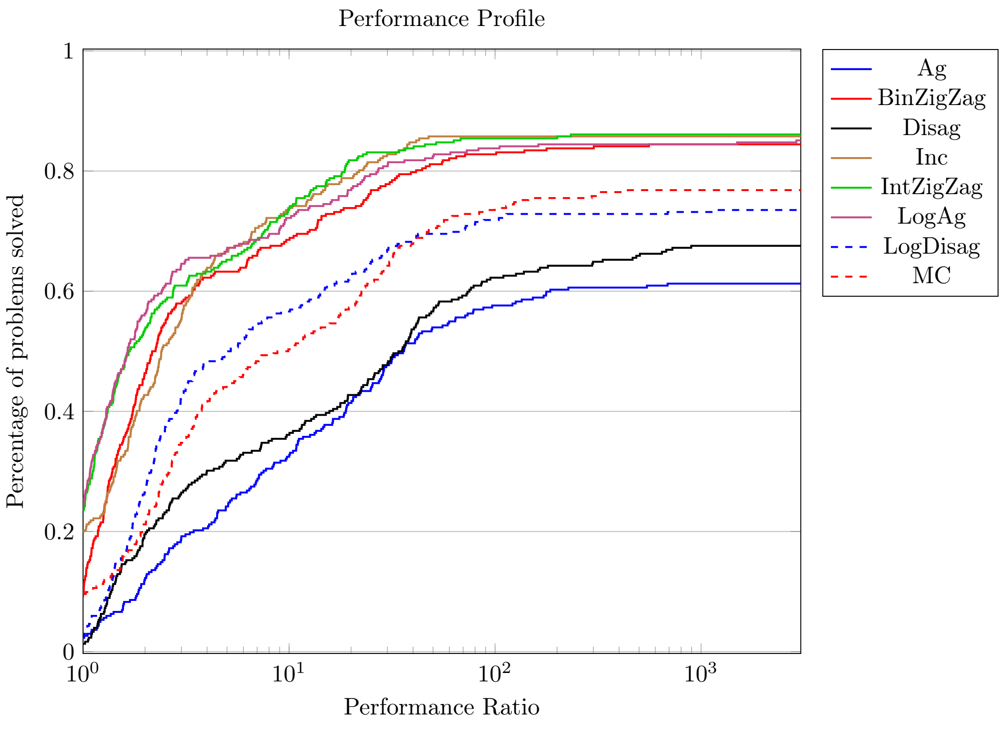
(14A) $\epsilon = 100$



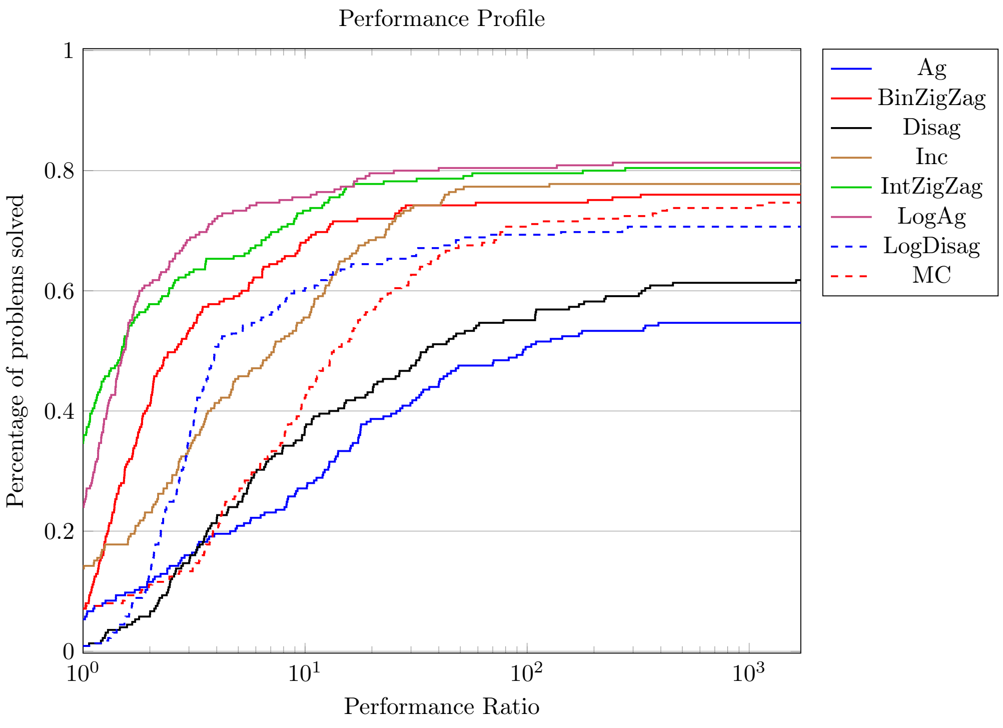
(14B) $\epsilon = 1$



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



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



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

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

TABLE 13. 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)	6.36	9.00	10.08	10.39	16.20	19.20
(LogDisag)	7.05	10.12	11.20	9.03	13.17	15.00
(Ag)	6.85	9.67	11.07	9.45	14.72	18.05
(LogAg)	6.59	8.68	9.32	8.40	11.64	12.61
(Inc)	**4.20**	**5.08**	**5.49**	**7.06**	**9.65**	**10.38**
(MC)	5.41	7.26	7.85	9.06	13.53	15.22
(BinZigZag)	5.78	7.26	7.79	7.92	10.82	11.84
(IntZigZag)	6.07	7.32	7.81	7.52	10.20	11.17
Error bound	$\epsilon = 10^{-2}$			$\epsilon = 10^{-4}$		
Time limit	100s	1000s	10000s	100s	1000s	10000s
(Disag)	22.73	47.32	63.27	23.36	42.43	52.71
(LogDisag)	19.39	33.11	39.76	17.65	27.50	31.49
(Ag)	18.90	34.31	42.68	24.56	51.02	84.71
(LogAg)	15.93	24.19	27.65	13.25	20.18	22.27
(Inc)	**14.66**	**22.41**	**24.51**	13.88	22.19	25.12
(MC)	19.20	34.59	43.31	18.55	33.96	41.19
(BinZigZag)	16.15	24.67	28.52	15.08	22.78	24.84
(IntZigZag)	15.78	24.11	27.29	**13.09**	**19.29**	**20.93**
Error bound	$\epsilon = 10^{-6}$					
Time limit	100s	1000s	10000s			
(Disag)	28.85	45.35	51.60			
(LogDisag)	17.90	21.92	22.17			
(Ag)	26.23	51.85	65.18			
(LogAg)	**10.07**	**10.81**	**10.81**			
(Inc)	18.28	28.13	28.97			
(MC)	28.02	46.04	52.80			
(BinZigZag)	12.80	14.47	14.47			
(IntZigZag)	12.01	12.84	12.84			

Table 14. 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)	5.86	7.89	8.37	9.15	12.97	14.11
(LogDisag)	6.55	9.18	9.90	7.82	10.22	10.57
(Ag)	6.40	8.66	9.44	8.23	11.52	12.66
(LogAg)	6.15	7.87	8.19	7.21	8.97	9.14
(Inc)	**3.80**	**4.42**	**4.58**	**5.91**	**7.13**	**7.21**
(MC)	4.93	6.27	6.53	7.86	10.46	10.83
(BinZigZag)	5.34	6.47	6.71	6.74	8.15	8.30
(IntZigZag)	5.63	6.56	6.76	6.36	7.74	7.90
Error bound	$\epsilon = 10^{-2}$			$\epsilon = 10^{-4}$		
Time limit	100s	1000s	10000s	100s	1000s	10000s
(Disag)	19.09	33.60	36.62	15.53	22.16	25.20
(LogDisag)	15.87	23.51	24.09	10.40	11.57	11.66
(Ag)	15.37	23.28	23.92	16.26	21.61	22.99
(LogAg)	12.80	16.77	16.77	6.48	6.95	6.95
(Inc)	**11.31**	**14.48**	**14.58**	6.92	7.42	7.42
(MC)	15.71	23.46	24.34	11.15	14.89	16.05
(BinZigZag)	13.02	17.09	17.15	8.08	8.48	8.48
(IntZigZag)	12.56	16.27	16.29	**6.45**	**6.74**	**6.79**
Error bound	$\epsilon = 10^{-6}$					
Time limit	100s	1000s	10000s			
(Disag)	25.47	35.90	39.37			
(LogDisag)	14.92	16.01	16.01			
(Ag)	22.88	38.94	40.63			
(LogAg)	**7.72**	**7.72**	**7.72**			
(Inc)	15.16	21.67	22.20			
(MC)	24.65	35.45	39.17			
(BinZigZag)	10.16	10.45	10.45			
(IntZigZag)	9.68	9.79	9.79			