A Review and Comparison of Solvers for Convex MINLP

Jan Kronqvist[a*], David E. Bernal[b], Andreas Lundell[c], and Ignacio E. Grossmann[b]

[a]Process Design and Systems Engineering, Åbo Akademi University, Åbo, Finland;
[b]Department of Chemical Engineering, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA;
[c]Mathematics and Statistics, Åbo Akademi University, Åbo, Finland;

June 3, 2018

Abstract

In this paper, we present a review of deterministic solvers for convex MINLP problems as well as a comprehensive comparison of a large selection of commonly available solvers. As a test set, we have used all MINLP instances classified as convex in the problem library MINLPLib, resulting in a test set of 366 convex MINLP instances. A summary of the most common methods for solving convex MINLP problems is given to better highlight the differences between the solvers. To show how the solvers perform on problems with different properties, we have divided the test set into subsets based on the integer relaxation gap, degree of nonlinearity, and the relative number of discrete variables. The results presented here provide guidelines on how well suited a specific solver or method is for particular types of MINLP problems.

Convex MINLP MINLP solver Solver comparison Numerical benchmark

1 Introduction

Mixed-integer nonlinear programming (MINLP) combines the modeling capabilities of mixed-integer linear programming (MILP) and nonlinear programming (NLP) into a versatile modeling framework. By using integer variables, it is possible to incorporate discrete decisions, *e.g.,* to choose between some specific options, into the optimization model. Furthermore, by using both linear and nonlinear functions it is possible to accurately model a variety of different phenomena, such as chemical reactions, separations, and material flow through a production facility. The versatile modelling capabilities of MINLP means there are a wide variety of real-world optimization problems that can be modeled as MINLP problems, *e.g.,* cancer treatment planning [35], design of water distribution networks [30], portfolio optimization [23], nuclear reactor core fuel reloading [110], process synthesis [63], pooling problems in the petrochemical industry [101], and production planning [112]. More of MINLP applications are described by, *e.g.,* [20, 29, 52, 118].

MINLP problems are, in general, considered as a "difficult" class of optimization problems. However, thanks to a great effort over the years, there has been significant progress in the field and there are several solvers available for MINLP problems available today, see [32]. Here we will focus on convex MINLP, which is a specific subclass with some desirable properties, *e.g.,* a convex MINLP problem is possible to decompose into a finite sequence of tractable subproblems. In recent years there has been significant progress within the field of MILP and NLP [3, 6] which is also reflected onto the field of MINLP since decomposition techniques for MINLP problems often rely on solving these types of subproblems. It is also possible to solve certain classes of nonconvex

*Corresponding author. Email: jan.kronqvist@abo.fi

MINLP problems, such as problems with signomial constraints, by reformulating them into convex MINLP problems [93, 94, 106, 108], further motivating the study of efficient methods for convex MINLP.

The intention of this paper is to give an overview of commonly available deterministic solvers for convex MINLP problems and to present a thorough numerical comparison of the most common solvers. Most optimization solvers are connected to one or more of the well-established modeling environments for MINLP optimization, such as, AIMMS [21], AMPL [55], and GAMS [31]. In recent years, there has also been a growing interest in optimization modeling in Python and Julia [18]; JuMP is a modeling environment for optimization embedded in Julia [42], and Pyomo is a similar environment in Python [68].

The solvers considered in the numerical comparison are AlphaECP, Antigone, AOA, BARON, BONMIN, COUENNE, DICOPT, Juniper, KNITRO, LINDO, Minotaur, Pajarito, SBB, SCIP, and SHOT. These solvers in the benchmark study were chosen based on criteria like availability, active development, and support for a file format available in MINLPLib [100]. Some of these are global solvers and are not limited to convex problems. However, most of the global solvers have some convexity identification techniques or manual strategy settings that can be set by the user to allow them to more efficiently deal with convex problems. The convex solvers can also often be used as heuristic methods without guarantee for finding the optimal solution for nonconvex MINLP problems.

In Section 2 the convex MINLP problem is defined and a general overview over the most common algorithms for such problems are given in Section 3. Most solvers in the comparison utilize one or more of these solution methods, as described in Section 4, which contains a summary of the solvers considered. Section 5 describes the benchmark in detail, and the numerical results are, finally, presented in Section 6.

2 Convex MINLP problem formulation

A convex MINLP problem can, without loss of generality, be written as

$$\min_{\mathbf{x},\mathbf{y}\in N\cap L\cap Y} \mathbf{c}_1^T\mathbf{x} + \mathbf{c}_2^T\mathbf{y}, \tag{P-MINLP}$$

where the sets N, L and Y are given by

$$\begin{aligned} N &= \{\mathbf{x}\in\mathbb{R}^n, \mathbf{y}\in\mathbb{R}^m \mid g_j(\mathbf{x},\mathbf{y}) \le 0 \quad \forall j = 1,\dots l\}, \\ L &= \{\mathbf{x}\in\mathbb{R}^n, \mathbf{y}\in\mathbb{R}^m \mid \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} \le \mathbf{b}\}, \\ Y &= \{\mathbf{y}\in\mathbb{Z}^m \mid \underline{y}_i \le y_i \le \overline{y}_i \quad \forall\, i = 1,2,\dots,m\}. \end{aligned} \tag{1}$$

where $L\cap Y$ is assumed to be a compact set. The upper and lower bounds on the integer variable y_i are denoted as $\overline{y}_i$ and $\underline{y}_i$ and are assumed to be finite, thus ensuring compactness of the set Y. Generally, problem (P-MINLP) is considered as convex if all the nonlinear functions g_j are convex in the variables $\mathbf{x}$, and the relaxed integer variables $\mathbf{y}$. There has recently been an interest in nonsmooth convex MINLP, and some solution techniques have been presented see *e.g.,* [44, 45]. However, most of the commonly available solvers only have guaranteed convergence for smooth problems and, therefore, we limit this study to this problem type where the nonlinear functions g_j are continuously differentiable.

3 Methods

This section describes the most commonly used algorithms for convex MINLP. The methods described are branch and bound, extended cutting plane, extended supporting hyperplane, outer approximation, generalized Benders decomposition and LP/NLP based branch and bound. This summary is not intended to give an in-depth analysis of the algorithms, but to better exemplify

the differences between the solvers. For a more detailed discussion about the algorithms see, *e.g.,* [10], [64], and [52].

3.1 Branch and bound

Branch and bound (BB) was first presented as a technique for solving MILP problems by [83]. A few years later it was noted by [38] that MINLP problems can be solved with a similar branch and bound approach, although the paper focused on linear problems. Solving convex MINLP problems with a BB approach was later studied by [66].

In the basic form, BB solves the MINLP problem by relaxing the integer restrictions of the original problem and solving continuous (convex) NLP relaxations. Solving an integer relaxations of problem (P-MINLP) results in a solution $(\mathbf{x}^k, \mathbf{y}^k)$, which provides a valid lower bound. If all components of $\mathbf{y}^k$ take on integer variables, then it is also an optimal solution to the MINLP problem. Otherwise, the continuous relaxation is divided (branched) into two new NLP subproblems by adding the constraints $y_i \leq \lfloor y_i^k \rfloor$ and $y_i \geq \lceil y_i^k \rceil$ to the relaxed problem. The branching variable y_i is a variable that takes on a fractional value and usually chosen based on some criteria, *e.g.,* the variable furthest away from an integer value. A new lower bound can be obtained by solving the new subproblems (child nodes), and in case one of the subproblems returns an integer solution it also provides a valid upper bound. The search procedure is often illustrated as a tree, where the nodes are connected to their parent node and represent the subproblems. If one of the nodes does not provide an integer solution, then it is branched into two new nodes creating two new subproblems. In case one of the nodes obtains an optimum worse than the upper bound or in case the subproblem is infeasible, then the node can be pruned off since the optimal solution cannot exist in that part of the search space. This approach of solving convex NLP problems in each node is often referred to as NLP-based branch and bound (NLP-BB).

Obtaining a tight integer relaxation is of great importance within BB to avoid extremely large search trees. [115] presented a branch and cut method for convex MINLP problems that uses cutting planes to strengthen the integer relaxation. Several techniques have been proposed for obtaining cuts to strengthen the integer relaxation for MINLP problems, *e.g.,* lift-and-project cuts [5, 77, 128], Gomory cuts [36, 61], and perspective cuts [56].

Compared to BB techniques for MILP problems, NLP-BB involves computationally more demanding subproblems; it is often not unusual to explore more than 100,000 nodes for a modest-sized problem! Techniques to efficiently integrate the NLP solver and not solving all subproblems to optimality have also been proposed by [28] and [87]. Another BB approach is to solve LP relaxations in the nodes and construct a polyhedral approximation of the nonlinear constraints. A polyhedral branch and cut technique, solving LP relaxations in the nodes, was presented by [117].

Many important details on BB has been left out, such as branching strategies. For more details on BB see, *e.g.,* [26, 53].

3.2 Extended cutting plane

The extended cutting plane (ECP) algorithm was first presented by [124], and can be seen as an extension of Kelley's cutting plane method for convex NLP problems presented by [75]. In its original form the ECP method is intended for convex MINLP problems, and by some modifications, given the name generalized alpha ECP (GAECP), it can be applied to pseudoconvex problems as shown by [125].

The ECP algorithm uses linearization of the nonlinear constraints to construct an iteratively improving polyhedral outer approximation of the set N. The trial solutions are obtained by solving the following MILP subproblems

$$(\mathbf{x}^{k+1}, \mathbf{y}^{k+1}) \in \underset{\mathbf{x},\mathbf{y} \in \hat{N}_k \cap L \cap Y}{\arg\min} \mathbf{c}_1^T \mathbf{x} + \mathbf{c}_2^T \mathbf{y}, \tag{MILP-k}$$

where the set $\hat{N}_k$ is given by

$$\hat{N}_k = \left\{ g_j(\mathbf{x}^i, \mathbf{y}^i) + \nabla g_j(\mathbf{x}^i, \mathbf{y}^i)^T \begin{bmatrix} \mathbf{x} - \mathbf{x}^i \\ \mathbf{y} - \mathbf{y}^i \end{bmatrix} \leq 0,\ \forall i = 1, 2 \dots k,\ j \in A_i \right\}. \tag{2}$$

Here A_i is an index set containing the indices of either the most violated or all violated constraints in iteration i. Set $\hat{N}_k$ is, thus, a polyhedral approximation of set N, constructed by first-order Taylor series expansions of the nonlinear constraints generated at the trial solutions $(\mathbf{x}^k, \mathbf{y}^k)$. The linearizations defining $\hat{N}_k$ is usually referred to as cutting planes since they cut off parts of the search space that cannot contain the optimal solution. Due to convexity, $N \subseteq \hat{N}_k$ and therefore, the solution of problem (MILP-k) provides a valid lower bound of problem (P-MINLP).

In the first iteration the set $\hat{N}_0$ can simply be defined as $\mathbb{R}^{n+m}$. New trial solutions are then obtained by solving subproblem (MILP-k), and the procedure is repeated until a trial solution satisfies the constraints within a given tolerance. Once a trial solution satisfies all nonlinear constraints it is also the optimal solution, since the solution was obtained by minimizing the objective within a set containing the entire feasible region. For more details on the ECP algorithm see, *e.g.,* [124] and [125].

3.3 Extended supporting hyperplane

The extended supporting hyperplane (ESH) algorithm was presented by [80] as an algorithm for solving convex MINLP problems. The ESH algorithm uses the same technique as the ECP algorithm for obtaining trial solutions, but uses a different technique for generating the polyhedral outer approximation $\hat{N}_k$. It has been observed that the cutting planes used to construct the polyhedral outer approximation in the ECP algorithm are, in general, not as tight as possible, see [80]. By using a simple procedure, the ESH algorithm is able to obtain supporting hyperplanes to set N at each iteration, and use these to construct a polyhedral outer approximation $\hat{N}_k$.

First, a strict interior point $(\mathbf{x}_{\text{int}}, \mathbf{y}_{\text{int}})$ is obtained by solving the following convex NLP problem

$$\begin{aligned} \min_{(\mathbf{x},\mathbf{y}) \in L, \mu \in \mathbb{R}} \quad & \mu \\ \text{s.t.} \quad & g_j(\mathbf{x}, \mathbf{y}) \leq \mu \quad \forall j = 1, 2, \dots, l. \end{aligned} \tag{3}$$

The interior point should preferably be as deep as possible within the interior of N, which is here approximated by minimizing the l_∞-norm of the nonlinear constraints.

Similar to the ECP algorithm, the trial solutions are obtained by solving problem (MILP-k), and they are now denoted as $\left(\mathbf{x}^k_{\text{MILP}}, \mathbf{y}^k_{\text{MILP}}\right)$. The trial solutions provide a valid lower bound on the optimal solution of problem (P-MINLP), however, they will not be used to construct the set $\hat{N}_k$.

To construct the polyhedral outer approximation, we define a new function F as the point-wise maximum of the nonlinear constraints, according to

$$F(\mathbf{x}, \mathbf{y}) = \max_j \left\{ g_j(\mathbf{x}, \mathbf{y}) \right\}. \tag{4}$$

A new sequence of points $\left(\mathbf{x}^k, \mathbf{y}^k\right)$ are now defined as

$$\begin{aligned} \mathbf{x}^k &= \lambda^k \mathbf{x}_{\text{int}} + (1 - \lambda^k) \mathbf{x}^k_{\text{MILP}}, \\ \mathbf{y}^k &= \lambda^k \mathbf{y}_{\text{int}} + (1 - \lambda^k) \mathbf{y}^k_{\text{MILP}}, \end{aligned} \tag{5}$$

where the interpolation parameters λ^k are chosen such that $F(\mathbf{x}^k, \mathbf{y}^k) = 0$. The interpolation parameters λ^k can be obtained by a simple one-dimensional root search. The points $\left(\mathbf{x}^k, \mathbf{y}^k\right)$ are now located on the boundary of the feasible region, and linearizing the active nonlinear constraints at this point results in supporting hyperplanes to the set N. The set $\hat{N}_k$ is, thus, constructed according to eq.(2) using the points $\left(\mathbf{x}^k, \mathbf{y}^k\right)$.

The ESH algorithm also uses a preprocessing step to obtain supporting hyperplanes of the set N by solving linear programming (LP) relaxations. The procedure of solving MILP subproblems

and generating supporting hyperplanes is repeated until a trial solution satisfies all nonlinear constraints. The tighter polyhedral outer approximation usually gives the ESH algorithm an advantage over the ECP algorithm. It has been shown in [45], that the ESH algorithm can also be successfully applied to nonsmooth MINLP problems with pseudoconvex constraint functions.

3.4 Outer approximation

The outer approximation (OA) method was first presented by [43], and some further properties for convex MINLP problems were presented by [50]. Some modifications of the OA method have been presented to handle nonconvex problems more efficiently, see, *e.g.,* [78, 121]. For more details on the basic convex approach discussed in this paper see, *e.g.,* [64].

OA is a decomposition technique, which obtains the optimal solution of the original problem by solving a sequence of MILP and NLP subproblems. Similar to both ECP and ESH, OA also constructs an iteratively improving polyhedral outer approximation $\hat{N}_k$ of the nonlinear feasible region defined by the set N. However, OA only uses the polyhedral approximation for choosing the integer combination $\mathbf{y}^k$, and the corresponding continuous variables $\mathbf{x}^k$ are chosen by solving a convex NLP subproblem.

In each iteration, the polyhedral outer approximation is used to construct problem (MILP-k), which is often referred to as the MILP master problem. A new integer combination $\mathbf{y}^k$ is then obtained by solving problem (MILP-k). Once the integer combination $\mathbf{y}^k$ is obtained, the following NLP subproblem is formed

$$\begin{aligned}(\mathbf{x}^k, \mathbf{y}^k) \in \underset{(\mathbf{x},\mathbf{y})\in N\cap L}{\arg\min}\ & \mathbf{c}_1^T\mathbf{x} + \mathbf{c}_2^T\mathbf{y} \\ \text{s.t.}\quad & \mathbf{y} = \mathbf{y}^k.\end{aligned} \tag{NLP-fixed}$$

If problem (NLP-fixed) is feasible, a valid upper bound can be obtained from the solution $(\mathbf{x}^k, \mathbf{y}^k)$, and the solution is used to improve the polyhedral approximation according to Eq. (2). The polyhedral outer approximation is updated by either linearizing all constraints or only the active constraints.

Problem (NLP-fixed) may also be infeasible in some iteration. If $\mathbf{y}^k$ is an infeasible integer combination, the corresponding continuous variables can be obtained by solving the following convex subproblem

$$\begin{aligned}(\mathbf{x}^k, \mathbf{y}^k, r^k) \in \underset{(\mathbf{x},\mathbf{y})\in L, r\in\mathbb{R}}{\arg\min}\ & r \\ \text{s.t.}\quad & \mathbf{y} = \mathbf{y}^k, \\ & g_j(\mathbf{x},\mathbf{y}) \le r \quad \forall j = 1, 2, \ldots, l,\end{aligned} \tag{NLP-feasibility}$$

which minimizes the constraint violation with respect to the l_∞-norm. The solution to problem (NLP-feasibility) does not provide a lower bound. However, using the infeasible solution $(\mathbf{x}^k, \mathbf{y}^k)$ to update the polyhedral outer approximation according to Eq. (2), ensures that the infeasible integer combination $\mathbf{y}^k$ cannot be obtained again by the MILP-master problem, *cf.* [50].

The OA algorithm is usually initiated by solving an integer relaxation of the MINLP problem, giving an initial lower bound and the solution can be used to construct the polyhedral approximation $\hat{N}_0$. It is also possible to use integer cuts to exclude specific integer combinations, as suggested by [43]. Solving the MILP master problems (MILP-k) provides a lower bound on the optimum, and the procedure is repeated until the upper and lower bound is within a given tolerance.

In general, OA results in tighter polyhedral outer approximations than the ECP algorithm, and may, therefore, require fewer iterations. However since each iteration is somewhat more computationally demanding, the two methods are difficult to compare directly in this way.

3.5 Generalized Benders decomposition

Generalized Benders decomposition (GBD) was first presented by [58] and is a generalization of Benders decomposition, a partitioning procedure for solving MILP problems [11]. As noted by

[109], GBD is closely related to OA and the main difference is the derivation of the master problem. In GBD, the master problem is projected onto the space defined by the integer variables and the master problem is, thus, only expressed in the integer variables. Here we will not present the full derivation of GBD, but use the same approach as [64] to derive the master problem. For more details on GBD see, *e.g.,* [65] or [52].

Given an integer combination $\mathbf{y}^k$, the corresponding continuous variables can be obtained by solving either of the problems (NLP-fixed) or (NLP-feasibility). If problem (NLP-fixed) is feasible, it provides a valid upper bound, as well as values for the continuous variables $\mathbf{x}^k$ and the optimal Lagrangean multipliers λ^k and μ^k. A valid cut is then given by

$$\mathbf{c}_2^T\mathbf{y} + \sum_{j=1}^{l} \lambda_j^k \nabla_{\mathbf{y}} g_j(\mathbf{x}^k, \mathbf{y}^k)^T(\mathbf{y} - \mathbf{y}^k) + (\mu^k)^T\mathbf{B}\mathbf{y} \leq \alpha, \tag{6}$$

where $\nabla_{\mathbf{y}}$ denotes the gradient with respect to the integer variables. Note that, the left-hand side of Eq. (6) is a first order Taylor series expansion of the Lagrangean function at the point $(\mathbf{x}^k, \mathbf{y}^k, \lambda^k, \mu^k)$ with respect to the $\mathbf{x}$ and $\mathbf{y}$ variables. The cut in Eq. (6) can be shown to be a surrogate constraint of the linearization in Eq. (2) in which the continuous variables $\mathbf{x}$ are projected out, *e.g.,* [109], and [64].

If problem (NLP-fixed) is infeasible with the integer combination $\mathbf{y}^k$, problem (NLP-feasibility) is solved to obtain the continuous variables $\mathbf{x}^k$ as well as the optimal multipliers λ^k and μ^k. A valid cut in the integer space is then given by,

$$\sum_{j=1}^{l} \lambda_j^k \left(g_j(\mathbf{x}^k, \mathbf{y}^k) + \nabla_{\mathbf{y}} g_j(\mathbf{x}^k, \mathbf{y}^k)^T(\mathbf{y} - \mathbf{y}^k)\right) + (\mu^k)^T\mathbf{B}\mathbf{y} \leq 0. \tag{7}$$

For more details on the cuts see, *e.g.,* [109]. The master problem for obtaining new integer combinations, is then given by

$$\begin{aligned}
\min_{\mathbf{y}\in Y, \alpha\in\mathbb{R}} \quad & \alpha \\
\text{s.t.} \quad & \mathbf{c}_2^T\mathbf{y} + \sum_{j=1}^{l} \lambda_j^k \nabla_{\mathbf{y}} g_j(\mathbf{x}^k, \mathbf{y}^k)^T(\mathbf{y} - \mathbf{y}^k) + (\mu^k)^T\mathbf{B}\mathbf{y} \leq \alpha, \ \forall k \in K_f \\
& \sum_{j=1}^{l} \lambda_j^k \left(g_j(\mathbf{x}^k, \mathbf{y}^k) + \nabla_{\mathbf{y}} g_j(\mathbf{x}^k, \mathbf{y}^k)^T(\mathbf{y} - \mathbf{y}^k)\right) \\
& \qquad + (\mu^k)^T\mathbf{B}\mathbf{y} \leq 0, \quad \forall k \in K\backslash K_f,
\end{aligned}$$

where K_f contains the indices of all iterations where problem (NLP-fixed) is feasible and the index set K contains all iterations. Solving the master problems provides a lower bound on the optimal solution and gives a new integer combination $\mathbf{y}^{k+1}$. The procedure is repeated until the upper and lower bounds are within the desired tolerance.

Since the cuts obtained by equations (6) and (7) can be viewed as surrogate cuts of the linear constraints included in the OA master problem, GBD generates weaker cuts than OA at each iteration and usually requires more iterations to solve a given problem. However, the master problems in GBD may be easier to solve since they contain fewer variables compared to OA and only one cut is added in each iteration.

A compromise between OA and GBD has been proposed by [109], where the continuous variables are classified into linear or nonlinear based on how they are involved in the original MINLP problem. By projecting out the nonlinear continuous variables, one can derive a Lagrangean cut similar as with GBD while still retaining the linear constraints involving continuous variables in the master problem. The given method has been coined as partial surrogate cuts (PSC), and as proved in [109], it results in a tighter linear relaxation compared to GBD while still only adding one cut per iteration.

3.6 LP/NLP based branch and bound

When solving a convex MINLP problem with either ECP, ESH, GBD or OA, most of the total solution time is usually spent on solving the MILP subproblems. The MILP problems are also quite similar in consecutive iterations since they only differ by a few linear constraints. To avoid constructing many similar MILP branch and bound trees, [109] presented a method which integrates OA within BB, called LP/NLP based branch and bound (LP/NLP-BB). The main idea is to only construct one branch and bound tree, where the MILP master problem is dynamically updated.

An initial polyhedral outer approximation is constructed by solving an integer relaxation and linearizing the constraints at the integer relaxed solution, as in OA. The polyhedral outer approximation is used to construct the first MILP master problem and branch and bound procedure is initiated, where an LP relaxation is solved in each node. Once an integer solution is obtained at a given node, the integer combination is used as in OA. If the NLP problem (NLP-fixed) with the given integer combination is feasible, it provides an upper bound and new linearizations are generated. If it is infeasible, new linearizations can be obtained by solving the feasibility problem (NLP-feasibility). The new linearizations are then added to all open nodes, and the LP relaxation is resolved for the node which returned the integer combination. The branch and bound procedure continues normally by solving LP relaxations, which now give a more accurate approximation of the nonlinear constraints. Here, the search must continue down each node until either the LP relaxation returns an integer solution that satisfies all nonlinear constraints, the LP relaxation obtains an objective value worse than the upper bound, or until the LP relaxation becomes infeasible. As in normal BB, a lower bound is provided by the lowest optimal solution of the LP relaxations in all open nodes, and the search continues until the upper and lower bounds are within a given tolerance. The LP/NLP-BB procedure, thus, only generates a single branch and bound tree, and is sometimes referred to as a single tree OA.

Numerical results have shown that LP/NLP-BB technique can results in significantly fewer nodes than the total number of nodes explored in the multiple MILP master problems in OA [43, 85]. Implementations of the LP/NLP-BB algorithm have shown promising results, *e.g.,* see [1, 24] or [96].

3.7 Primal heuristics

Primal heuristics is a common term for algorithms and techniques intended to obtain good feasible solutions quickly. The use of primal heuristics began in the field of MILP. For instance, [49] claimed that primal heuristics were one of the most important improvements in MILP solvers within the last decade. In recent years, there has also been an interest in primal heuristics for MINLP problems and several algorithms have been proposed for this task. Such algorithms are, *e.g.,* undercover [17], feasibility pump [25], rounding heuristics [16], and the center-cut algorithm [81]. Another technique for obtaining feasible solutions in solvers based on ECP, ESH or OA, is to check the alternative solutions in the solution pool provided by the MILP solver [80]. A detailed summary of several primal heuristics for MINLP problems is given by [15] and [39].

Finding a good feasible solution to an MINLP problem can improve the performance of MINLP solvers, as shown by the numerical results in [13, 15]. Knowing a good feasible solution can, *e.g.,* reduce the size of the search tree in BB based solvers, and provide a tight upper bound. Obtaining a tight upper bound is especially important in solvers based on the ECP or ESH algorithm, because, in their basic form neither algorithm will obtain a feasible solution before the very last iteration.

3.8 Preprocessing

Preprocessing includes various techniques for modifying the problem into a form more favorable for the actual solver. The preprocessing procedures can result in tighter relaxations or reduce the problem size. [10] classified MINLP presolving techniques into two major categories: housekeeping and reformulations. Housekeeping includes techniques such as bound tightening and removal

of redundant constraints. Reformulations in the preprocessing can include techniques such as improvement of coefficients in the constraints and disaggregation of constraints.

There are two main approaches for tightening the variable bounds, feasibility based bound tightening [9, 114], and optimization-based bound tightening [89]. Feasibility based bound tightening analyzes the constraints sequentially to improve the variable bounds, whereas optimization-based bound tightening solves a sequence of relaxed problems where each variable is maximized and minimized to obtain optimal bounds.

By reformulating the original problem, it is in some cases possible to obtain significantly tighter relaxations. Within MILP it is well known that different problem formulations can result in a tighter or weaker integer relaxation. The uncapacitated facility location problem is an example where disaggregation of some constraints leads to a tighter integer relaxation [126]. Similar techniques can also be used to obtain tighter integer relaxations for MINLP problems. Some types of nonlinear constraints can also be disaggregated to obtain a lifted reformulation of the problem, where the nonlinear constraint is split into several constraints by the introduction of new variables. Such lifted reformulations were first proposed by [117], where it was shown that a lifted reformulation results in tighter polyhedral outer approximations. In a recent paper by [82], it was shown that the performance of several MINLP solvers, based on ECP, ESH, and OA, could be drastically improved by utilizing a reformulation technique based on lifting. Lifted reformulations of MINLP problems have also been studied by [69], and [92]. Some further reformulation techniques for MINLP problems are also presented in [88].

4 Solvers

This section is intended as an introduction to commonly available MINLP solvers, and to describe their main properties. Most of the solvers are not based on a single "pure" algorithm but they combine several techniques and ideas to improve their performance. On top of this, MINLP solver technology has evolved from two more mature and different ends, NLP and MILP solvers. This results in most of the solvers for MINLP relying on different MILP and NLP solvers. Among the MILP solvers, the most recognized commercial solvers are CPLEX [73], Gurobi [67], and XPRESS [48], while the solvers GLPK [97] and CBC [54], the last one from the COIN-OR initiative [91], are among the most recognized open-source solvers, all of them implementing an arsenal of methods within a branch and cut framework. In the NLP case, solvers like CONOPT [41], SNOPT [59], and Knitro [34] are well-known commercial options, and IPOPT [122] is a well-known open-source solver, also part of the COIN-OR initiative. There exists more variability in the algorithms behind NLP solvers, *e.g.,* CONOPT implements a Generalized Reduced Gradient (GRG) method, while IPOPT and Knitro use the interior-point method, and SNOPT uses a sequential quadratic programming (SQP) approach, see [19] for a review in NLP.

In this section, we only mention the main features of the solvers, and for more details see the references given in the solver sections. A summary of solvers and software for MINLP problems was previously also given by [32]. The available solvers are implemented in a variety of programming languages, some as stand-alone libraries accessible from algebraic modeling software like GAMS, AMPL, and AIMMS. Other solvers have been implemented directly in the same programming languages as their modeling systems, *e.g.,* MATLAB, Python-Pyomo, Julia-JuMP. The solvers used in the numerical comparison are listed in alphabetical order below.

AlphaECP (**A**lpha **E**xtended **C**utting **P**lane) is a solver based on the ECP algorithm developed by T. Westerlund's research group at Åbo Akademi University in Finland, and implemented for GAMS by T. Lastusilta. By using the GAECP algorithm [125] the solver also has guaranteed convergence for pseudoconvex MINLP problems. The solver mainly solves a sequence of MILP subproblems, but to speed-up convergence, it occasionally solves convex NLP subproblems with fixed integer variables. To improve the capabilities of handling nonconvex problems, the algorithm also employs some heuristic techniques, described by [84]. As subsolvers, AlphaECP can utilize all the NLP and MILP solvers available in GAMS. An important feature of the solvers is the technique

used for terminating the MILP subsolver prematurely presented by [125], usually resulting in only a small portion of the MILP subproblems to be solved to optimality and a significant reduction in the total solution time. More details on the solver are found at `www.gams.com/latest/docs/S_ALPHAECP.html`.

ANTIGONE (**A**lgorithms for co**NT**inuous / **I**nteger **G**lobal **O**ptimization of **N**onlinear **E**quations) is a global optimization solver available in GAMS. The solver was developed by R. Misener and C.A. Floudas at Princeton University. ANTIGONE uses reformulations and decomposes the nonconvex constraints into constant, linear, quadratic, signomial, linear fractional, exponential, and other general nonconvex terms. Convex relaxations are then generated for the decomposed nonconvex terms and the relaxations are solved in a branch and cut framework. As a global solver, ANTIGONE is not limited to only convex problems and is able to solve a variety of nonconvex problems. ANTIGONE uses local solvers for finding feasible solutions and uses MILP relaxations to obtain valid lower bounds, resulting in a sequence of relaxations converging to the global optimum [102]. The solver can use CONOPT and SNOPT as NLP subsolver, and CPLEX as subsolver for MILP relaxations. More details on the solver are found at `www.gams.com/latest/docs/S_ANTIGONE.html`.

AOA (**A**IMMS **O**uter **A**pproximation) is a system module implemented in the AIMMS language originally developed by Paragon Decision Technology [72]. As the name suggests, the solver is based on OA and implements both normal OA and the LP/NLP-BB methods. The latter is the recommended one for convex problems and generates linearizations as lazy constraints utilizing MILP solver callbacks. AOA is implemented as a "white box solver", meaning that it allows the users to fully customize the algorithm. To improve the performance and its capabilities for solving nonconvex problems, AOA may use nonlinear preprocessing and a multi-start technique. More details on this module can be found at `aimms.com/english/developers/resources/solvers/aoa/`.

BARON (**B**ranch and **R**educe **O**ptimization **N**avigator) is a global optimization solver developed by N.V. Sahinidis at Carnegie Mellon University (partially at the University of Illinois) and M. Tawarmalani at Purdue University [111, 117]. The solver is available in many different environments such as AIMMS, AMPL, GAMS, JuMP, MATLAB and Pyomo [113]. The solver uses a polyhedral branch and bound technique, and thus, solves LP relaxations in the BB nodes. However, BARON also uses MILP relaxations as described by [127] and [76]. Nonconvex problems are handled by generating convex underestimators (and concave overestimators) in combination with a spatial branch and bound technique. The solver utilizes automatic reformulations and convexity identification to decompose nonconvex functions into simpler functions with known convex (or concave) relaxations. The reformulations can also result in tighter lifted polyhedral outer approximations as shown by [117]. BARON also uses advanced bound tightening and range reduction techniques to reduce the search space in combination with other techniques, such as local search techniques. In GAMS, BARON is able to use all the available NLP solvers for subproblems and CBC, CPLEX, and XPRESS as LP/MILP subsolver. More information on BARON can be found at `http://www.minlp.com/baron`.

BONMIN (**B**asic **O**pen-source **N**onlinear **M**ixed **In**teger Programming) is an open-source solver for MINLP problems developed in collaboration between Carnegie Mellon University and IBM Research, lead by P. Bonami, as a part of the COIN-OR initiative. The solver implements several algorithms, and the user is able to choose between NLP-BB, NLP/LP-BB, OA, feasibility pump, OA based branch and cut, and a hybrid approach. Some computational results, as well as detailed descriptions of the main algorithms, are given by [24]. BONMIN has an interface to Pyomo, JuMP, AMPL, GAMS, and MATLAB (through the OPTI Toolbox, *cf.* [37]). There is also a commercial version of the solver, called BONMINH, in which the NLP subsolver IPOPT may utilize additional linear solvers [71]. LP and MILP relaxations can be solved with both CBC

and CPLEX. More details of the solver are given by [22], and detailed description of the project and solver can be found at `projects.coin-or.org/Bonmin`.

Couenne (**C**onvex **O**ver and **U**nder **E**nvelopes for **N**onlinear **E**stimation) is a global optimization open-source solver for MINLP problems. It was developed as part of the COIN-OR initiative in collaboration between Carnegie Mellon University and IBM Research, and the project was led by P. Belotti. The solver implements an LB based spatial branch and bound technique as its main algorithm, in addition to bound reduction techniques and primal heuristics [8]. Couenne is closely integrated with BONMIN and extends it by routines for calculating valid linear outer approximations of nonconvex constraints. For solving NLP subproblems, Couenne uses IPOPT as a subsolver. Couenne is accessible in GAMS, Pyomo, JuMP, and AMPL. A solver manual is provided in [7], and more information about Couenne can be found at `projects.coin-or.org/Couenne`.

DICOPT (**Di**screte **C**ontinuous **Opt**imizer) is a solver based on the OA method for MINLP, developed by the group of I.E. Grossmann at Carnegie Mellon University. This solver implements the equality relaxation and augmented penalty methods in combination with OA [121]. In the equality relaxation, the nonlinear equality constraints are relaxed as inequalities using dual information of the constraints, and the augmented penalty method relaxes the linearizations with slack variables which are penalized in the objective of the MILP master problem of OA. Both methods are intended as heuristics for nonconvex MINLP problems, although if the equality constraints relax as convex inequalities, the methods become rigorous as the slacks of the augmented penalty are driven to zero. DICOPT is only available in GAMS, and it can use any available MILP or NLP solvers to solve the subproblems. Recently, a feasibility pump algorithm was implemented as a primal heuristic for convex MINLP problems to improve the solver's performance [13]. More information about DICOPT can be found at `www.gams.com/latest/docs/S_DICOPT.html`.

Juniper is an open-source MINLP solver implemented in Julia. It is developed by O. Kröger, C. Coffrin, H. Hijazi, and H. Nagarajan at Los Alamos National Laboratory. The solver implements an NLP-BB method with branching heuristics, primal heuristics, *e.g.,* the feasibility pump, and parallelization options available in Julia [79]. The NLP subproblems can be solved *e.g.,* with IPOPT. More information about Juniper can be found at `github.com/lanl-ansi/Juniper.jl`.

Knitro is a commercial optimization software currently developed by Artelys [34]. Knitro implements an interior-point/active-set algorithm for dealing with nonlinear continuous problems. Although it can be used as an NLP solver, it also and solves MINLP problems by a branch and bound approach. The solver is available in, *e.g.,* AIMMS, AMPL, JuMP, and GAMS. More information about Knitro is found at
`www.artelys.com/en/optimization-tools/knitro`.

LINDO is a global solver developed by LINDO Systems Inc. available in both GAMS, the LINDO environment LINGO, as an add-in for Microsoft Excel, and MATLAB. LINDO includes solvers for LP, quadratic programming (QP), NLP and conic programming [90]. For mixed-integer problems, LINDO uses a branch and cut approach and is able to utilize both linear and nonlinear subsolvers [74]. To fully use the global optimization capabilities in GAMS, LINDO requires the NLP solver CONOPT. For nonconvex problems, the solver uses reformulations and convex relaxations within a BB framework. The solver is able to recognize convex quadratic terms, and an option for turning off the global search strategies, *i.e.,* for convex problems, is available. More information about the solver is found at `www.lindo.com`.

Minotaur (**Mixed-I**nteger **N**onlinear **O**ptimization **T**oolkit: **A**lgorithms, **U**nderestimators, and **R**elaxations) is an open-source toolkit for solving MINLP problems developed in collaboration between Argonne National Laboratory, Indian Institute of Technology Bombay, and the University of Wisconsin-Madison. The goals of Minotaur are to (1) provide efficient MINLP solver, (2)

implement a variety of algorithms in a common framework, (3) provide flexibility for developing new algorithms for exploiting special structures, and (4) reduce the burden of developing new algorithms by providing a common software infrastructure [96]. Currently, Minotaur has two main approaches for convex MINLP based on the NLP-BB and LP/NLP-BB algorithms. Minotaur is able to use both filterSQP and IPOPT as NLP subsolvers and CBC, Gurobi, or CPLEX as LP subsolver. Minotaur is available as a standalone solver, but it has also an AMPL interface. Details about the Minotaur project can be found at `wiki.mcs.anl.gov/minotaur/`.

Pajarito is an open-source solver for Mixed-Integer Conic Programming (MICP) implemented in Julia by M. Lubin and J.P. Vielma from the Massachusetts Institute of Technology. Contrary to most of the other solvers presented in this manuscript, Pajarito can use a conic problem formulation by a conic outer approximation algorithm. The solver implements conic OA as well as classic OA and LP/NLP-BB algorithms. Using Disciplined Convex Programming (DCP) Pajarito can access lifted reformulations, which leads to tighter approximations of the original feasible region when projected back into the original variables [92]. Pajarito is currently accessible through Julia using problem inputs in both JuMP and DCP [119]. As MILP solvers it can use any that are available through JuMP; *e.g.,* CPLEX, Gurobi, and CBC; and for the subproblems, it can use either specialized conic solvers, *e.g.,* MOSEK [4], or NLP solvers, *e.g.,* IPOPT. More details about the solver can be found at `github.com/JuliaOpt/Pajarito.jl`.

SBB (**S**imple **B**ranch-and-**B**ound) is a solver in GAMS based on NLP-BB, developed by ARKI Consulting and Development A/S. The solver can use any NLP solver in GAMS for solving the integer relaxations. The solver uses GAMS Branch-Cut-and-Heuristic facility [57], to obtain cutting planes and for primal heuristics. More details of the solver are given at `www.gams.com/latest/docs/S_SBB.html`.

SCIP (**S**olving **C**onstraint **I**nteger **P**rograms) was originally developed by T. Acterberg at the Zuse Institut Berlin as a general framework based on branching for constraint and mixed-integer programming, *cf.* [2]. The solver is intended to be modular and it utilizes plugins to make it easy to modify. SCIP was extended by [120] to also solve convex and nonconvex MINLP problems utilizing spatial branch and bound in combination with OA and primal heuristics. SCIP is open source and free for academic use, and it is available both as a stand-alone solver and with interfaces to GAMS, Pyomo, JuMP, and MATLAB through the OPTI Toolbox. More information about SCIP can be found in the manual [60] or at `http://scip.zib.de`.

SHOT (**S**upporting **H**yperplane **O**ptimization **T**oolkit) is an open-source solver for convex MINLP developed by A. Lundell, J. Kronqvist and T. Westerlund at Åbo Akademi University. The solver utilizes polyhedral outer approximations, generated mainly by the ESH method, and iteratively constructs an equivalent MILP problem. The solver is closely integrated with the MILP solvers CPLEX and Gurobi, and uses a single-tree approach similar to the LP/NLP-BB technique. The supporting hyperplanes are dynamically added to the search tree by utilizing callbacks and lazy constraints, enabling the MILP solver to incorporate the new linearizations without generating a new branch and bound tree. The integration with the MILP solver enables SHOT to fully benefit from it with regards to, *e.g.,* cut generating procedures, advanced node selection, and branching techniques. SHOT also includes the functionality to solve MIQP subproblems. To obtain an upper bound to the MINLP problem SHOT utilizes several primal heuristics, including solving fixed NLP problems. SHOT is available as a stand-alone solver, but can also be integrated with GAMS, in which case SHOT can use any of its NLP solvers. A basic version of SHOT is also available for Wolfram Mathematica [95]. SHOT cannot use the single tree approach with CBC as MILP solver or in the Mathematica version, and in this case, solves a sequence of MILP relaxations in combination with the primal heuristics. More information about the solver is available at `http://www.github.com/coin-or/shot`.

Besides the solvers mentioned above, there are a few others solvers capable of solving convex MINLP problems that the authors are aware of. It should be noted that the solvers left out of the numerical comparison are not necessarily inferior compared to the other solvers. Some of these solvers could not read the problem formats available on MINLPLib, and were therefore automatically left out from the comparison.

bnb is a MATLAB implementation of NLP-BB by K. Kuipers at the University of Groningen. The solver uses the fmincon routine in MATLAB's Optimization Toolbox for solving the integer relaxed subproblems. The MATLAB code for the solver can be downloaded from `www.mathworks.com/matlabcentral/fileexchange/95-bnb`.

FICO Xpress Solver is a solver currently developed by FICO [47], and is available as both standalone binaries or as a FICO Xpress-MOSEL module. The solver is based on their NLP solver using successive linear programming (SLP) [107]. The solver includes two main algorithms as well as some heuristics. The first algorithm is based on NLP-BB where the integer relaxations are solved using the SLP solver, and the second algorithm is a version of OA. The solver also utilizes some other techniques described in [47], to improve its performance for MINLP problems. More information about FICO and its solvers can be found at `http://www.fico.com/en/products/fico-xpress-optimization`.

FilMINT is an MINLP solver developed by K. Abhishek, S. Leyffer and J. Linderoth based on the NLP/LP-BB algorithm [1]. The solver is built on top of the MILP solver MINTO [104] and uses filterSQP [51] for solving NLP relaxations. By utilizing functionality in MINTO, FilMINT is able to combine the NLP/LP-BB algorithm with features frequently used by MILP solvers, such as, cut generation procedures, primal heuristics, and enhanced branching and node selection rules. There is an AMPL interface available for FilMINT, and for more details, we refer to [1].

fminconset is an implementation of NLP-BB in MATLAB by I Solberg. The NLP subproblems are solved with MATLAB's fmincon routine in the Optimization Toolbox. The solver is available to download from `www.mathworks.com/matlabcentral/fileexchange/96-fminconset`.

GAECP (**G**eneralized **A**lpha **E**xtended **C**utting **P**lane) is a solver based on the GAECP algorithm [125] developed by T. Westerlund. The solver also uses supporting hyperplanes as in the ESH algorithm and is able to guarantee convergence for MINLP problems with nonsmooth pseudoconvex functions. The solver is described in detail in [123].

MILANO (**M**ixed-**I**nteger **L**inear and **N**onlinear **O**ptimizer) is a MATLAB based MINLP solver developed by H. Y. Benson at Drexel University. There are two versions of the solver available, one uses an NLP-BB technique and the other is based on OA. The NLP-BB technique version uses an interior point NLP solver with warm-starting capabilities described in [12]. The solver can be downloaded from `http://www.pages.drexel.edu/~hvb22/milano`.

MindtPy (**M**ixed-**I**nteger **N**onlinear **D**ecomposition **T**oolbox in **Py**omo) is an open-source software framework implemented in Python for Pyomo by the research group of Ignacio Grossmann at Carnegie Mellon University. This toolbox implements the ECP, GBD, and OA algorithms, together with primal heuristics, all relying on the decomposition and iterative solution of MINLP problems. It relies on Pyomo to solve the resulting MILP and NLP subproblems, allowing any of the solvers compatible with Pyomo to be used with MindtPy [14]. The toolbox is available in the following repository `github.com/bernalde/pyomo/tree/mindtpy`.

MINLP_BB was developed by S. Leyffer and R. Fletcher as a general solver for MINLP problems [86]. The solver is based on NLP-BB and uses the filterSQP for solving the integer relaxations. There is an interface to AMPL and the solver can also be used in MATLAB through the TOMLAB optimization environment [70]. More information about the solver is available at `wiki.mcs.anl.gov/leyffer`.

MISQP (**M**ixed-**I**nteger **S**equential **Q**uadratic **P**rogramming) is a solver based on a modified sequential quadratic programming (SQP) algorithm for MINLP problems presented by [46]. The solver is developed by K. Schittkowski's research group and the University of Bayreuth. MISQP is intended for problems where function evaluations may be expensive, *e.g.,* where some function values are obtained by running a simulation. There is an interface in MATLAB through TOMLAB as well as a standalone Fortran interface. A more detailed description of the solver is available at `http://tomwiki.com/MISQP`.

Muriqui is an open-source MINLP solver developed by W. Melo, M. Fampa, and F. Raupp, recently presented by [99]. The solver has several algorithms implemented, *e.g.,* ECP, ESH, OA, NLP/LP-BB, and NLP-BB, as well as some heuristics approaches [98]. For solving MILP and NLP subproblems, Muriqui can use CPLEX, Gurobi, Xpress [48], Mosek [4], Glpk [97], IPOPT and Knitro. Muriqui is written in C++ and can be integrated with AMPL. More information about the solver is available at `http://www.wendelmelo.net/software`.

There are a few other deterministic solvers that the authors are aware of, capable of handling convex MINLP problems but mainly focusing on nonconvex MINLP. These solvers are: Decogo (**DECO**mposition-based **G**lobal **O**ptimizer; [106]), POD (**P**iecewise convex relaxation, **O**uter-approximation, and **D**ynamic discretization; [103]), LaGO (**L**agrangian **G**lobal **O**ptimizer; [105]). For more details on nonconvex MINLP see, *e.g.,* [89, 116] and [53].

5 Benchmark details

The intention of the forthcoming two sections is to compare some of the convex MINLP solvers mentioned in the previous section by applying them on a comprehensive set of test problems. There are some benchmarks available in literature, *e.g.,* [1, 27, 80, 84]. However, these are limited to only a few of the solvers considered here or used a smaller set of test problems. The goal here is to give a comprehensive up-to-date comparison of both open-source and commercial solvers available in different environments. The main interest has been to study how the solvers perform on a standard desktop computer. All the benchmarks were performed on a Linux-based PC with an Intel Xeon 3.6 GHz processor with four physical cores (and which can process eight threads at once) and 32 GB memory. We have allowed the solvers to use a maximum of eight threads to replicate a real-world situation where one tries to solve the problems with all the available resources.

In the comparison, we have included three versions of BONMIN: BONMIN-OA which is based on OA, BONMIN-BB which is based on NLP-BB, and BONMIN-HYB which is a variant of the LP/NLP-BB algorithm. We have also included two versions of Minotaur: Minotaur-QG which is based on the LP/NLP-BB algorithm, and Minotaur-BB which is based on NLP-BB. The different versions were included since they represent different approaches for solving the MINLP problems, and the results vary significantly. The solvers in the comparison are implemented in and used from different environments (GAMS, AIMMS, and Julia/JuMP), and the subsolvers available may vary. Where possible, we have tried to use CONOPT and IPOPT(H) as NLP solver, and CPLEX as (MI)LP solver. A list of the solvers and subsolvers used are given in Table 1.

The termination criteria used with the solvers were the absolute and relative objective gap between the upper and lower objective bounds. The absolute objective gap was set to 10^{-3} and the limit for the relative gap was set to 0.1%. These termination criteria are supported by all the GAMS solver. However, the AIMMS and JuMP solvers did not support termination based

Table 1: The table shows which subsolvers were used with each solver, and on which platform the solver was run on.

MINLP solver	Subsolver used		Platform
	MILP/LP	NLP	
AlphaECP 2.10.06	CPLEX 12.8	CONOPT 3.17G	GAMS 25.0.3
Antigone 1.1	CPLEX 12.8	CONOPT 3.17G	GAMS 25.0.3
AOA Convex	CPLEX 12.8	CONOPT 3.14V	AIMMS 4.53.1
BARON 17.10.16	CPLEX 12.8	CONOPT 3.17G	GAMS 25.0.3
BONMINH 1.8	CPLEX 12.8	IPOPTH 3.12	GAMS 25.0.3
Couenne 0.5	CPLEX 12.8	IPOPTH 3.12	GAMS 25.0.3
DICOPT 2	CPLEX 12.8	CONOPT 3.17G	GAMS 25.0.3
Juniper 0.2.0	CPLEX 12.8	IPOPT 3.12.1	JuMP 0.18.1
Knitro 10.3	CPLEX 12.8	-	GAMS 25.0.3
Lindo 11.0	CPLEX 12.8	CONOPT 3.17G	GAMS 25.0.3
Minotaur 05-21-2018	CPLEX 12.6.3	filterSQP20010817	
Pajarito 0.5.1	CPLEX 12.8	IPOPT 3.12.1	JuMP 0.18.1
SBB	CPLEX 12.8	CONOPT 3	GAMS 25.0.3
SCIP 5.0	CPLEX 12.8	IPOPTH 3.12	GAMS 25.0.3
SHOT 05-21-2018	CPLEX 12.6.3	CONOPT 3.17G	GAMS 25.0.3

on the absolute gap. To make sure that the solvers did not terminate prematurely due to other built-in termination criteria and to avoid clear solver failures, some specific solver options were given; these are listed in Appendix B. Except for these, default settings were used for all solvers. Furthermore, a time limit of 900 seconds was also used with all solvers. Even with the 15-minute time limit, the total running time for the experiments was more than two weeks.

5.1 Problem sets

The problems considered here are all from the problem library MINLPLib [100], which currently consists of 1534 instances. These instances originate from several different sources and applications as indicated in the library. Out of the 1534 instances, we have chosen all problems that satisfy the following criteria: classified as convex, containing at least one discrete variable, and contain some nonlinearity (either in the objective function or in the constraints). There were in total 366 instances that satisfied the given criteria, and these constitute our master benchmark test set. Some statistics of the problems are available in Table 2

The problems in MINLPLib represent a variety of different types of optimization problems with different properties such as the number of variables, number of discrete variables, and number of nonlinear terms. Some of the problems also represent different formulations of the same problems, *e.g.,* some problems are written with both the big-M and convex hull formulation of disjunctions. It is therefore of interest to compare the solvers not only on the entire test set but also compare them on different subsets of problems with specific properties. We have partitioned the test set into groups, representing both integer and nonlinear properties, to compare both the solvers and algorithms for the different types of problems. The following criteria were used to partition the test problems into subsets:

Integer relaxation gap. By solving an integer relaxation of the MINLP problem and comparing the optimal objective value of the relaxed problem with the actual optimal objective value, we are able to determine the integer relaxation gap. To avoid differences due to scaling, we use a relative integer relaxation gap calculated as

$$\text{Relative integer relaxation gap} = \frac{|z^* - \bar{z}|}{\max\{|z^*|, 0.001\}} \cdot 100\%, \tag{8}$$

Table 2: Statistics of the convex MINLP instances used in the benchmark

objective function type	problem count
linear objective	274
quadratic objective	66
general nonlinear objective	26

	minimum	mean	maximum
number of discrete variables	2	93	1500
number of variables	3	913	107,223
number of constraints	0	1110	108,217
number of nonlinear constraints	0	12	112
number of nonlinear variables	1	129	45211

where z^* denotes the optimal objective value and $\bar{z}$ denotes to optimum of the integer relaxation. The integer relaxation gap varies significantly for the test problems: some instances have a gap larger than 1000% and for some instances it is smaller than 1%. Based on the gap, given by eq. (8), we have divided the test problems into two subsets: Problems with a large gap ($> 50\%$) and problems with a small gap ($< 50\%$). According to this classification, there are 151 problems with a large gap (average gap 190%) and 215 with a small gap (average gap 7.2%).

Nonlinearity. Some of the test problems are almost linear with only a few nonlinear terms, whereas some test problems are nonlinear in each variable. The test problems are thus classified based on the following nonlinearity measure

$$\text{Degree of nonlinearity} = \frac{n_{\text{nonlin}}}{n_{\text{tot}}} \cdot 100\%, \tag{9}$$

where n_{nonlin} is the number of variables involved in a nonlinear term and n_{tot} is the total number of variables. The test problems are divided into the following two categories based on the nonlinearity measure: Problems with high degree of nonlinearity ($> 50\%$), and problems with low degree of nonlinearity ($< 50\%$). The set with high degree of nonlinearity contains 133 problems with an average nonlinearity measure of 91%, while the set with low degree of nonlinearity contains 233 problems with an average nonlinearity measure of 14%.

Discrete density. The number of discrete variables also varies significantly in the test problems some problems contain only a few discrete variables, while others contain only discrete variables. To avoid a division based mainly on the problem size, we have chosen to divide the problems based on the following measure

$$\text{Discrete density} = \frac{n_{\text{int}} + n_{\text{bin}}}{n_{\text{tot}}} \cdot 100\%. \tag{10}$$

Here n_{int} and n_{bin} are the number of integer and binary variables, and n_{tot} is the total number of variables. Again the test problems are divided into two subsets: problems with a high discrete density ($> 50\%$) and problems with a low discrete density ($< 50\%$). The first category contains 151 problems with an average discrete density of 85%, and the second category contains 215 problems with an average density of 27%.

A list of the problems in each category is given in Appendix A, which also shows the integer relaxation gap, degree of nonlinearity, and discrete density for each test problem.

5.2 Reporting

All the results were analyzed using PAVER [33], which is a tool for comparing the performance of optimization solvers and analyzing the quality of the obtained solutions. The reports generated by PAVER, as well as all the results obtained by the individual solvers are available at `http://minlpcomparison.github.io`.

The parameters used for generating the reports are also available within the reports. A comment must be made regarding the choice of the parameter `gaptol`, which was set to the value $1.002 \cdot 10^{-3}$ instead of the value used as termination criteria ($1 \cdot 10^{-3} = 0.1\%$). The small perturbation is needed due to differences in how the relative gap is calculated by the solvers. Some of the solvers calculate the relative gap by dividing the gap by the lower bound, whereas others divide by the smallest absolute value of either the upper or lower bound. For example, BARON and ANTIGONE would, without the small perturbation, seem to terminate prematurely on a large number of instances and these would all be marked as failed by PAVER.

PAVER also calculate so-called **virtual best** and **virtual worst solvers**. The virtual best solver is the best (in our graphs the fastest) successful solver selected for each individual problem instance, and the virtual worst is then the slowest for each instance. These solvers provide a good comparison for how good an individual solver is compared to all solvers.

Since MINLPLib also provides a list of known optimal objective values, as well as upper and lower objective bounds, PAVER is able to compare the obtained solutions by the known bounds in MINLPLib. PAVER is, thus, also able to calculate the so-called primal gap, *i.e.,* the difference between the obtained solution and the best-known integer solution, which can be used to analyze the quality of the obtained solutions. For example, there are cases where the solver returns the optimal solution, but it has not been able to verify optimality within the time limit. PAVER also uses known objective bounds available in MINLPLib to check whether the solvers obtained correct solutions and bounds for the test problems.

6 Results

The results are presented using solution profiles showing the number of individual problems that a solver is able to solve as a function of time. Note that the profiles do not represent the cumulative solution time, but shows how many individual problems the solvers can solve within a specific time. We have not used performance profiles where the time is normalized with respect to best solver [40] since these are not necessarily good for comparing several solvers as noted by [62].

In all solution profiles in this section, we have chosen to divide the solvers into two categories to make the solution profiles more easily readable. The solvers are divided into MILP decomposition based solvers and BB based solvers. The division is not completely straightforward since some of the solvers could fit into both categories. However, the division is only intended to make it easier to read the results. The solvers classified as **MILP decomposition based solvers** are AlphaECP, ANTIGONE, AOA, BONMIN-OA, DICOPT, Minotaur-QG, Pajarito, and SHOT. Solvers classified as **BB based solvers** are BARON, BONMIN-BB, COUENNE, BONMIN-HYB, Juniper, Knitro, LINDO, Minotaur-BB, SBB, and SCIP. The time scales are also divided into two parts to better highlight differences between the solvers, and it is *(1)* linear in the first 10 seconds, and *(2)* logarithmic between 10 and 900 seconds. In each plot, the solvers in the nonactive group are indicated with thin gray lines, while the others are as shown in the respective legends. The same line style is used for a specific solver in all figures. In the right margin of each profile, the solvers are ranked according to the number of solved problems (as indicated within parenthesis). The virtual best and virtual worst solvers are shown in the figures as the top and bottom gray lines, and the region between them is shaded.

Figures 1 and 2 show the solution profiles when applying the solvers on the complete set of test problems. As mentioned, the solution profiles indicate the number of problems that have been solved by the individual solvers as a function of time. Out of the branch and bound based solvers, BARON is able to solve the most instances within the time limit followed by SCIP and

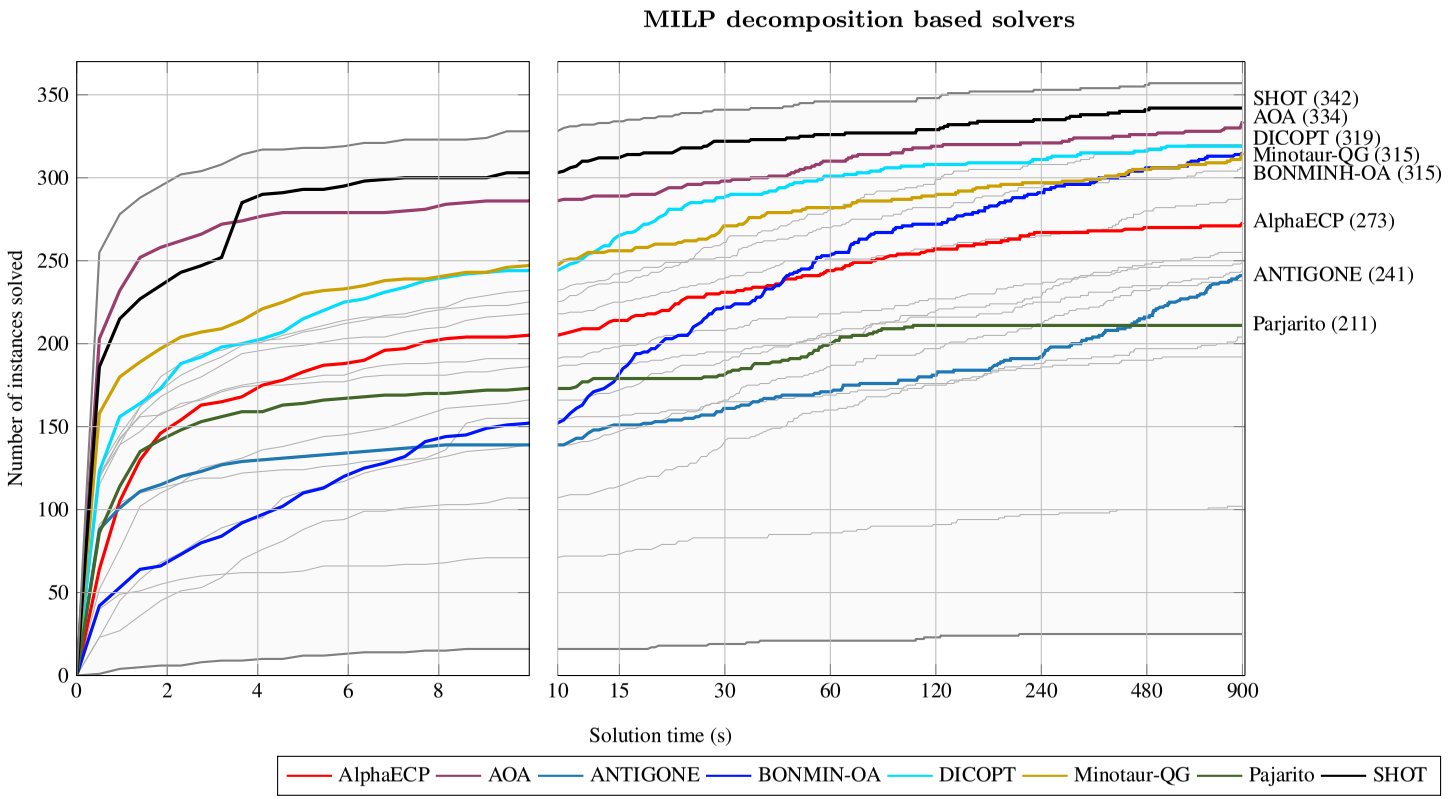

Figure 1: The solution profile indicates the number of solved convex MINLP instances in MINLPLib [100] as a function of time. A problem is regarded as solved if the absolute objective gap, as calculated by PAVER [33], is $\leq 0.1\%$. The border lines on top/below the shaded area indicates the virtual best/worst solver.

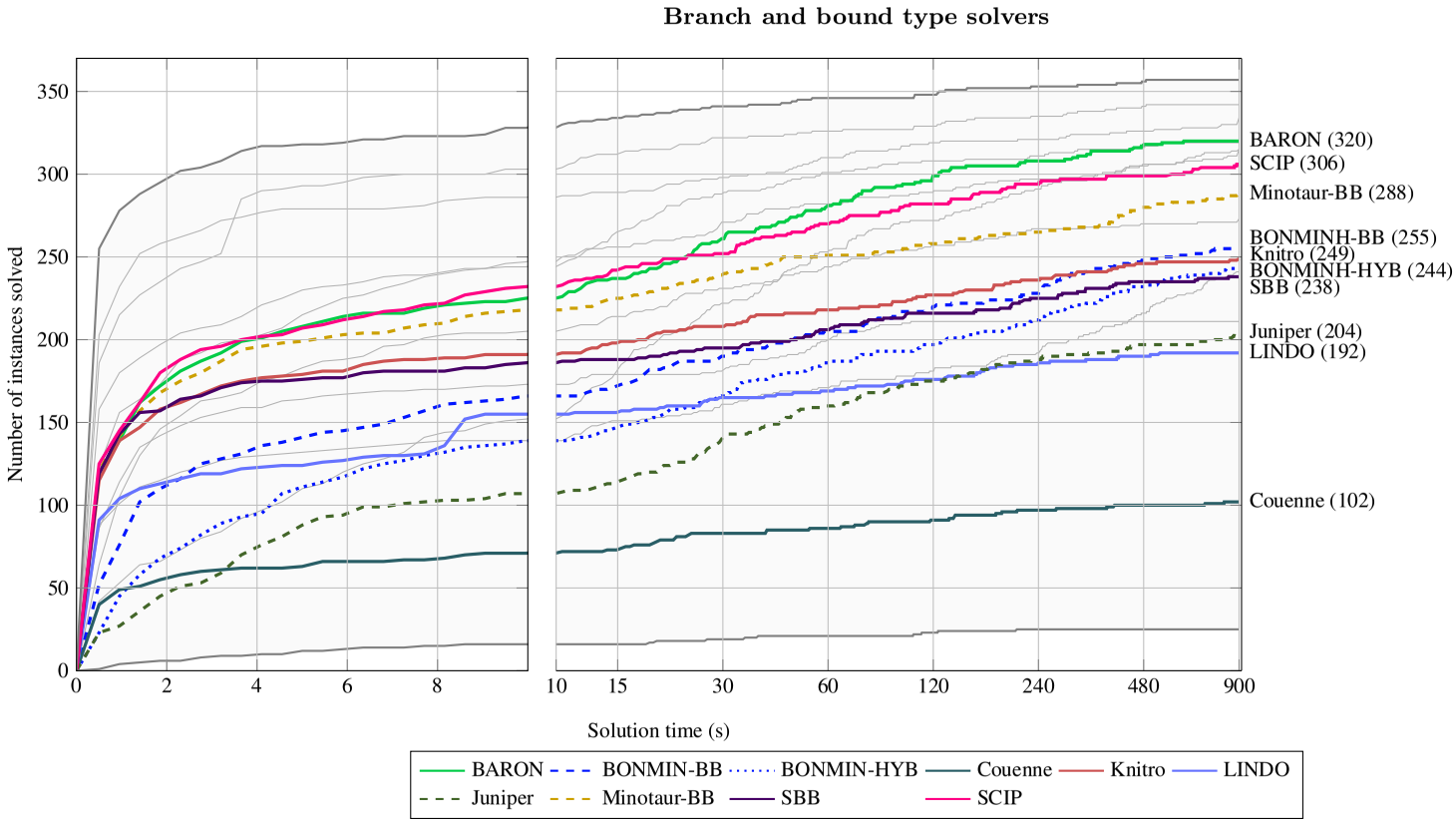

Figure 2: The solution profile indicates the number of solved convex MINLP instances in MINLPLib [100] as a function of time. A problem is regarded as solved if the absolute objective gap, as calculated by PAVER [33], is $\leq 0.1\%$. The border lines on top/below the shaded area indicates the virtual best/worst solver.

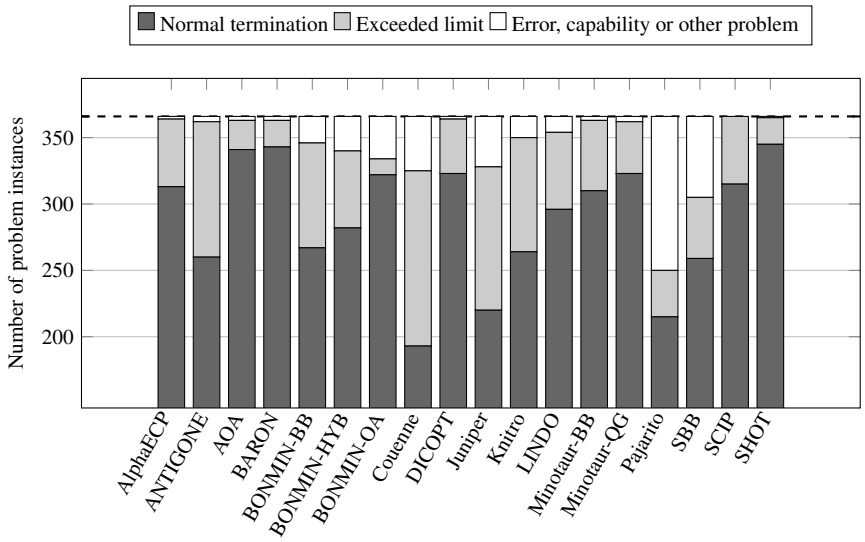

Figure 3: The solution status returned from the solvers.

Minotaur-BB. SHOT is able to solve the most problems out of the MILP decomposition based solvers, followed by AOA and DICOPT. SHOT and AOA are overall the fastest solvers for the test set, and they manage to solve 342(SHOT) and 334(AOA) of the 366 test problems within the time limit. The virtual best solver is able to solve 357 of the problems whereas the virtual worst only manages to solve 25 problems. The virtual best and worst solvers, thus, show a huge spread in the solvers' performance for different problems and highlight the importance of choosing a solver well suited for the problem type.

Figure 3 presents statistics regarding the termination of the solvers, *e.g.,* how many errors and timeouts occurred. These values are as reported by the solver, but verified by PAVER by comparing the solutions returned to known values or bounds, and checking if the solver fulfilled the time limit. If some solution returned by a solver is wrong, these instances are given the status error; the number of such instances per solver is reported in Figure 4. Such failure may *e.g.,* be due to numerical issues within the solver. Figure 5 shows the number of problems where the solver was able to obtain a solution within 0.1% and 1% of the best-known solution. The figure shows that none of the solvers was able to obtain a solution within 1% of the best-known solution for all of the problems, given the 900 second time limit. For example, BARON was able to obtain a solution within 1% of the optimum for 352 problems and SHOT obtained such a solution for 350 problems.

The number of instances solved to a relative objective gap, *i.e.,* difference between the upper and lower bound, of 0.1% and 1%, per solver is shown in Figure 6. By comparing Figures 5 and 6, it can be observed that some of the solvers able to obtain a solution within 0.1% of the optimum to significantly more problems than they are able to verify as optimal. For example, AlphaECP seems to be struggling with obtaining a tight lower bound for some of the problems, since it is able to obtain solutions within 0.1% of the optimum in 333 problems, but only verified optimality to a 0.1% gap for 273 instances.

Since it may be difficult to draw more detailed conclusions from the results in Figures 1 and 2, the next sections consider subsets of test problems with specific properties. A summary of the results for the different subsets is given in section 6.5.

6.1 Impact of the integer relaxation gap

In this section, we consider two types of problems: problems with a large integer relaxation gap, and problems with a small integer relaxation gap. Figure 7 shows the solution profiles of the solvers

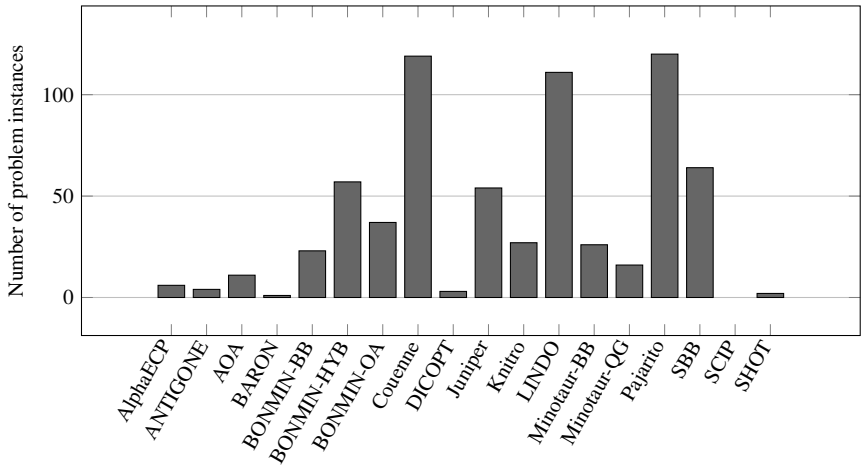

Figure 4: The number of solutions per solver flagged as error by PAVER. Most often, the cause is that the returned solution is not within the bounds provided in MINLPLib.

for the problems with a large gap, and Figure 8 shows the solution profiles for the problems with a small gap. By comparing the figures, there is a clear difference for the solvers based on a BB approach. The solvers utilizing a BB type algorithm clearly perform better on the problems with a small gap compared to the problems with a large integer relaxation gap. For example, Knitro is one of the most efficient solvers for the problems with a small gap, both in terms of speed and number of solved problems, whereas it is out-performed by several solvers for the problems with a large gap.

The NLP-BB based solvers, BONMIN-BB, Minotaur-BB, and SBB solve significantly fewer of the problems with a large gap than the solvers based on either an ECP, ESH or OA (AlphaECP, BONMIN-OA, DICOPT, and SHOT). Overall, the MILP decomposition based solvers seem to be less affected by the integer relaxation gap then the BB based solvers.

6.2 Impact of nonlinearity

The problem types considered in this section are problems with a high and low degree of nonlinearity, and the results are shown in Figures 9 and 10. Several of the solvers use linearizations to approximate the nonlinear functions in some steps of the solution procedure, whereas solvers using an NLP-BB approach directly treats the nonlinearity. Most of the solvers utilizing linearizations perform significantly better on the problems with a low degree of nonlinearity. For the problems with a low degree of nonlinearity, BARON, BONMIN-OA, and DICOPT are among the most efficient ones in terms of both speed and number of problems solved. However, for the problems with a high degree of nonlinearity they are outperformed by the NLP-BB based solvers Minotaur-BB, BONMIN-BB, and Knitro.

SHOT and the NLP/LP-BB based solver Minotaur-QG have quite similar behavior for both types of problems and perform quite well in both categories. Both of the solvers rely on linearizations of the nonlinear constraints and one would, thus, expect them to be negatively affected by the degree of nonlinearity. AOA, Minotaur-QG and SHOT all use a quite similar single-tree approach where NLP subproblems are solved in some of the nodes, which might help them to cope with problems with a high degree of nonlinearity.

The NLP-BB based solvers seem to be most affected by the degree of nonlinearity. For problems with a high degree of nonlinearity they performed overall well, and for the problems with a low degree of nonlinearity they did not perform as well in comparison with the other solvers.

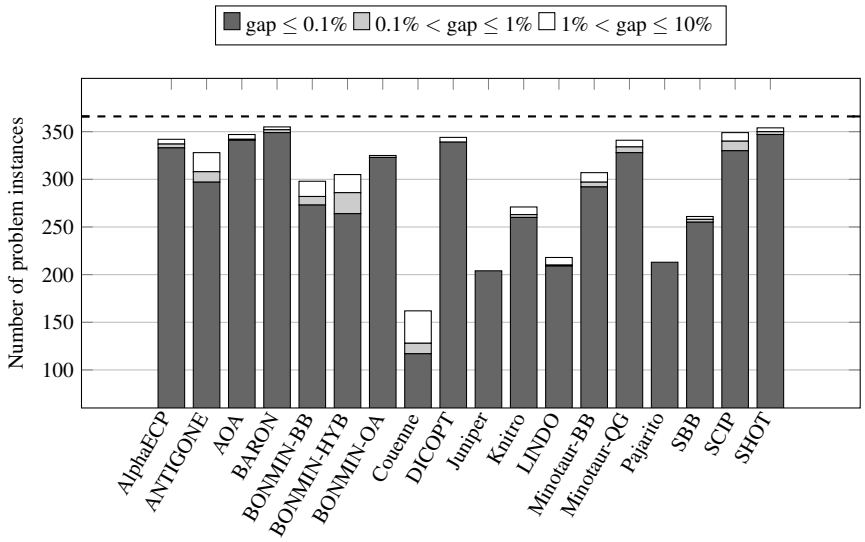

Figure 5: The number of instances in the benchmark with a solution within 0.1%, 1% and 10% of the best known optimal value.

6.3 Impact of discrete density

Finally, we compare how the solvers are affected by the percentage of discrete variables, *i.e.,* integer, and binary variables. Figures 11 and 12 show how the solvers perform for problems with high and low discrete density.

The MILP decomposition based solvers perform similar for both types of problems, and no conclusions can be drawn from the results. However, again there is a clear difference for the NLP-BB based solvers. Surprisingly, both BONMIN-BB, Knitro, and Minotaur-BB performed better, with respect to the other solvers, on the set of problems with a high discrete density.

Both BARON and SHOT perform well on both sets of test problems, and they perform somewhat better on the problems with a low discrete density. The OA approach seems to be well suited for the problems with a low discrete density, where DICOPT is one of the most efficient solvers and BONMIN-OA also manages to solve a large portion of the problems.

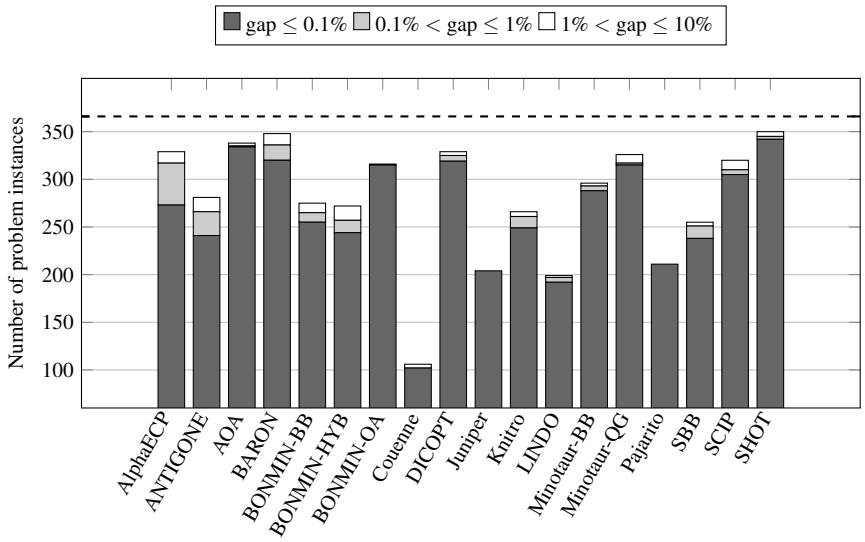

Figure 6: The number of instances in the benchmark with an objective gap of 0.1%, 1% and 10%.

6.4 Impact of preprocessing techniques

To illustrate the benefits of using preprocessing procedures on convex MINLP problems, we have solved all the problems using BARON with default preprocessing and with reduced preprocessing. BARON was chosen for this experiments since it probably has the most advanced preprocessing of all the considered solvers, and with the default setting it is able to solve a large portion of the test problems. BARON does not only uses bound tightening and range reduction in advance of the solution procedure but also in the branch and bound nodes. Reducing the bound tightening and range reduction procedures, thus, affects the entire solution procedure. However, the intention here is mainly to illustrate the benefits of using such preprocessing techniques.

The problems are first solved using default settings and with reduced preprocessing. For the test with reduced preprocessing we have disabled the linear- and nonlinear-feasibility-based range reduction (`LBTTDO=0`), not allowed any local searches in the preprocessing (`NumLoc=0`) and disabled the optimality based tightening (`OBTTDo=0`). More information about these settings is found in [113].

Figure 13 shows that performance is strongly affected by the preprocessing techniques mentioned earlier. With the reduced preprocessing, BARON was able to solve 65 fewer problems within the time limit. Overall the results clearly demonstrate the benefits of using well integrated preprocessing techniques.

6.5 Summary of the results

How the solvers are affected by the integer relaxation gap, degree of nonlinearity and discrete density is summarized in Table 3. The table shows the number of problems solved within each category as well as an indicator of how the solvers' performance was affected by the specific properties. The performance indicator shows how the performance of a solver is affected by the problem properties with respect to the other solvers. If a solver clearly performed better, with respect to speed and number of solved problems, in a category it is indicated by '+', and similarly '–' indicates that solver performed worse for that category of problems. If the performance is similar within both categories it is indicated by '≈'. These indicators were obtained by carefully analyzing the performance profiles, and are not intended as a grade of the solver but to show how it is affected by different problem properties. The results presented in Table 3 indicates that BB based solvers seem to be more affected by the problem properties considered here compared to the MILP decomposition based solvers.

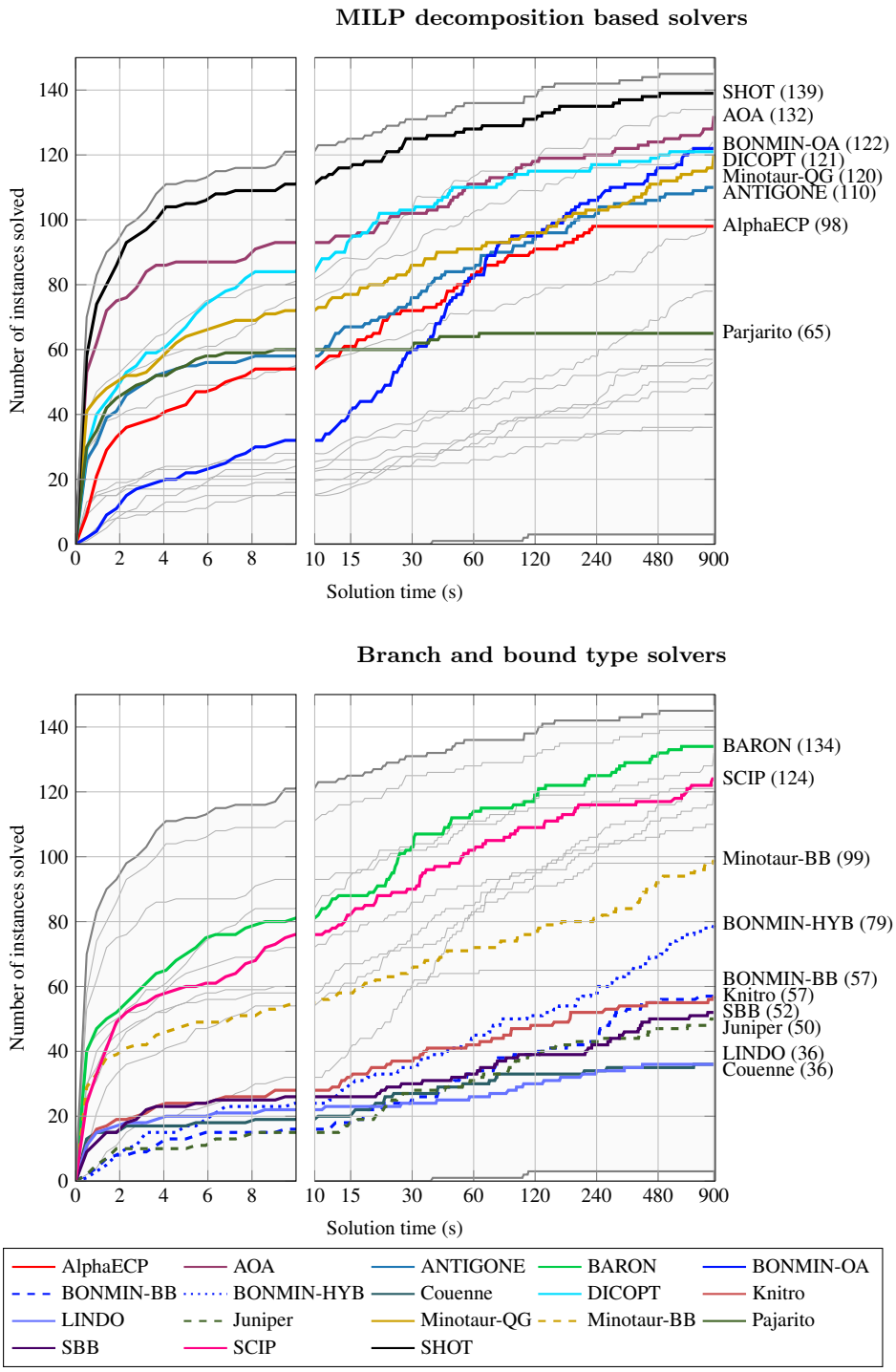

Figure 7: The solution profiles for problem instances with a high integer relaxation gap as indicated in Appendix A.

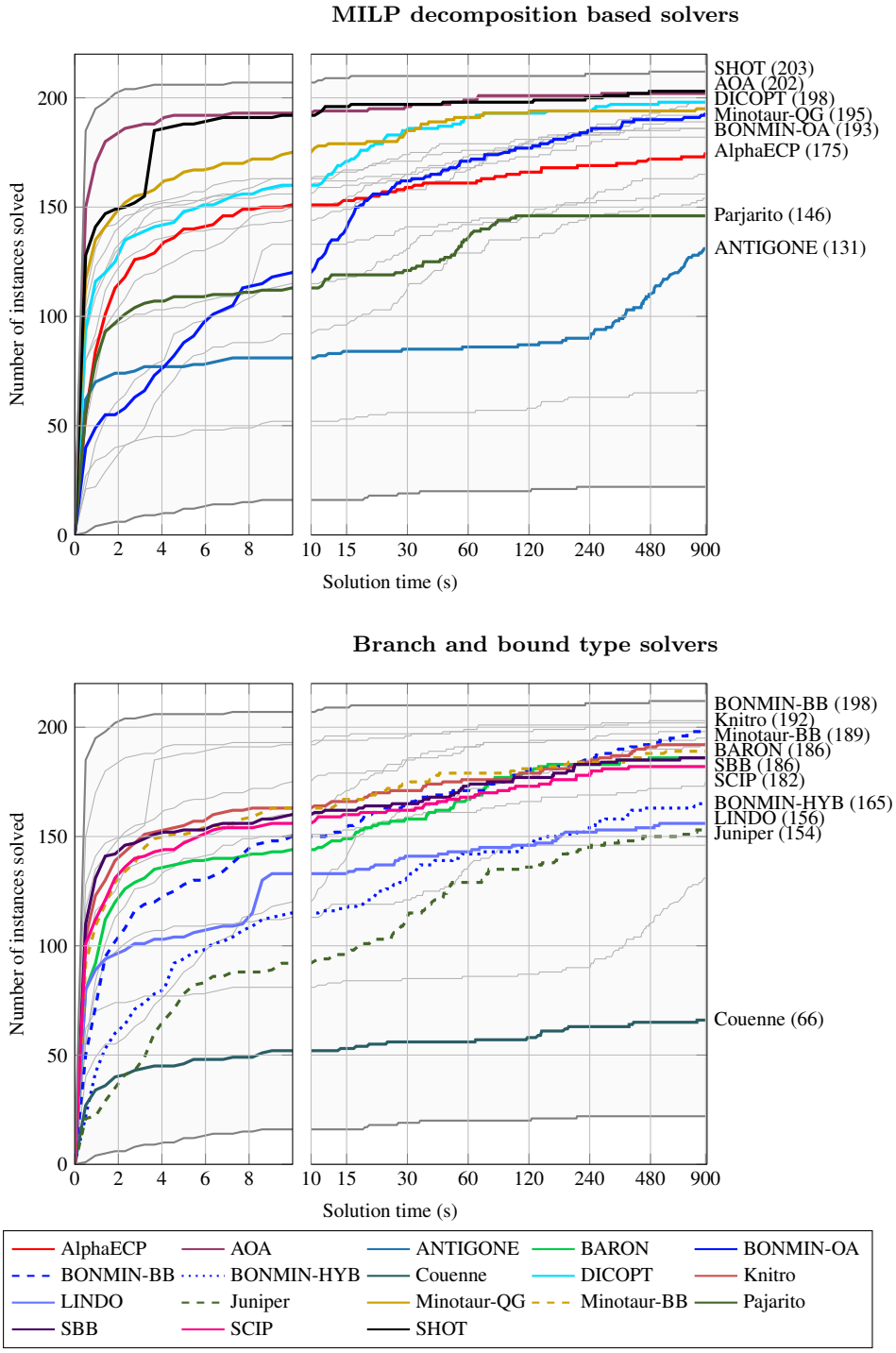

Figure 8: The solution profiles for problem instances with a low integer relaxation gap as indicated in Appendix A.

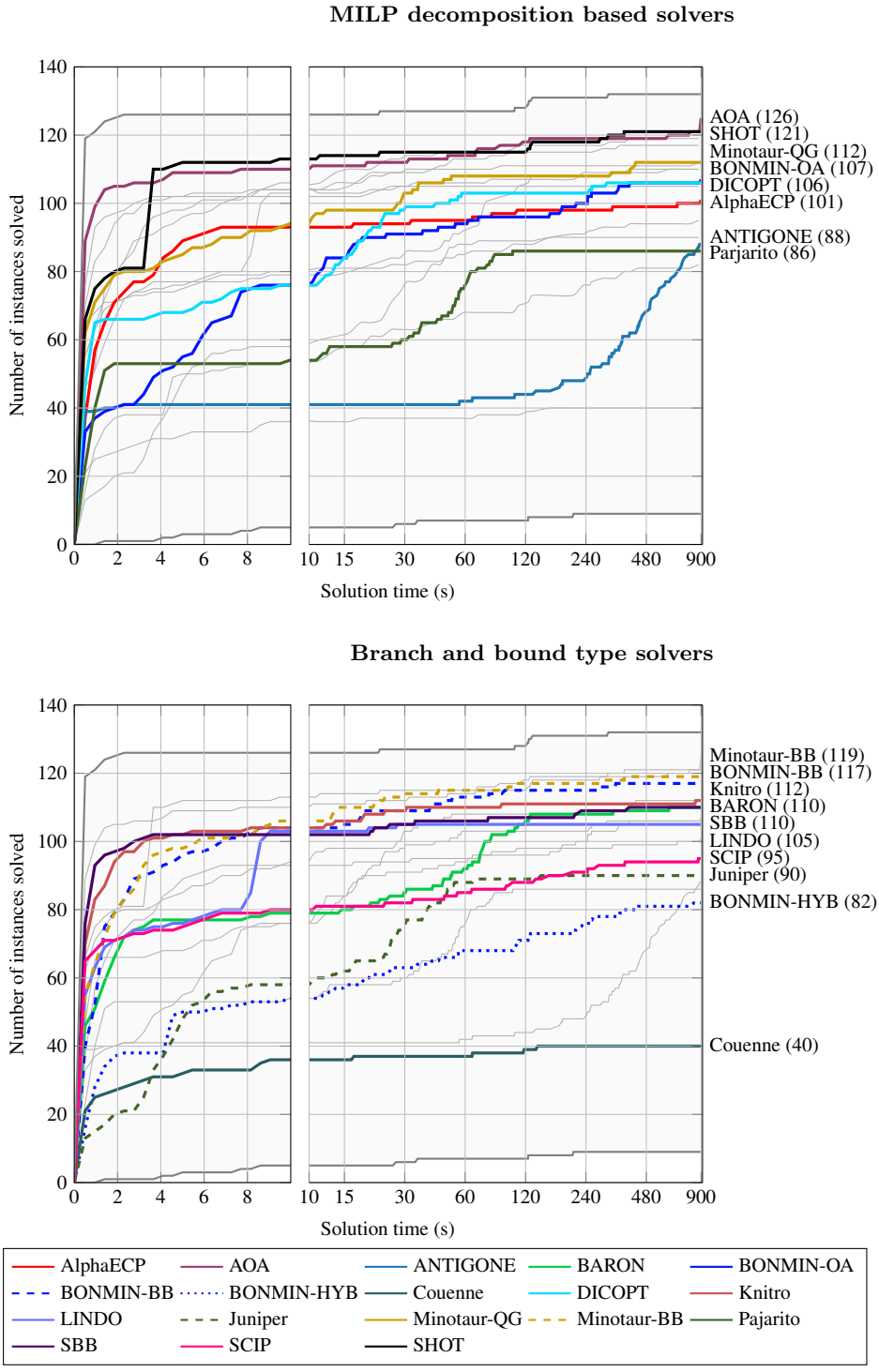

Figure 9: The solution profiles for problem instances with a high level of nonlinear variables as indicated in Appendix A.

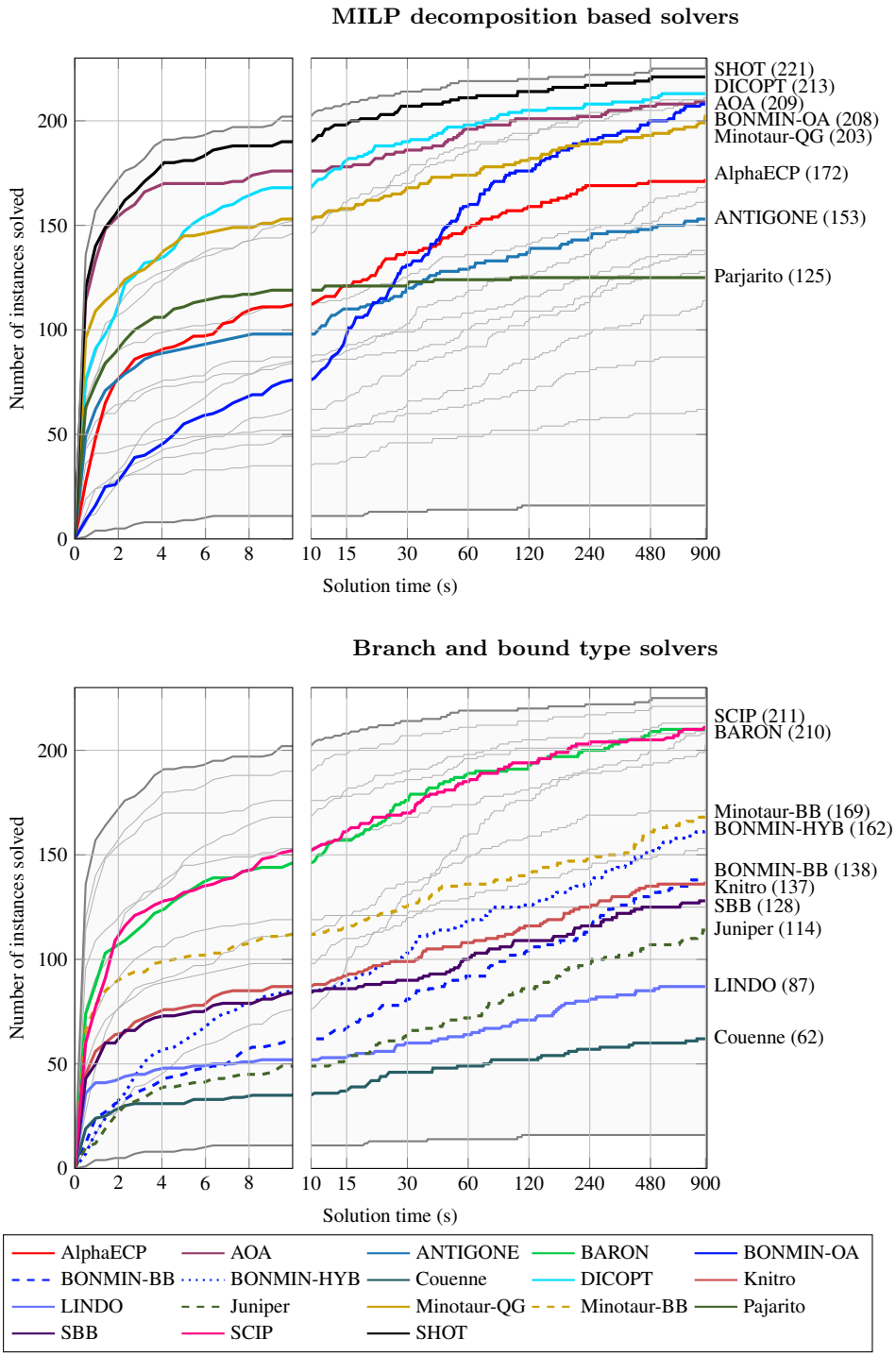

Figure 10: The solution profiles for problem instances with a low level of nonlinear variables as indicated in Appendix A.

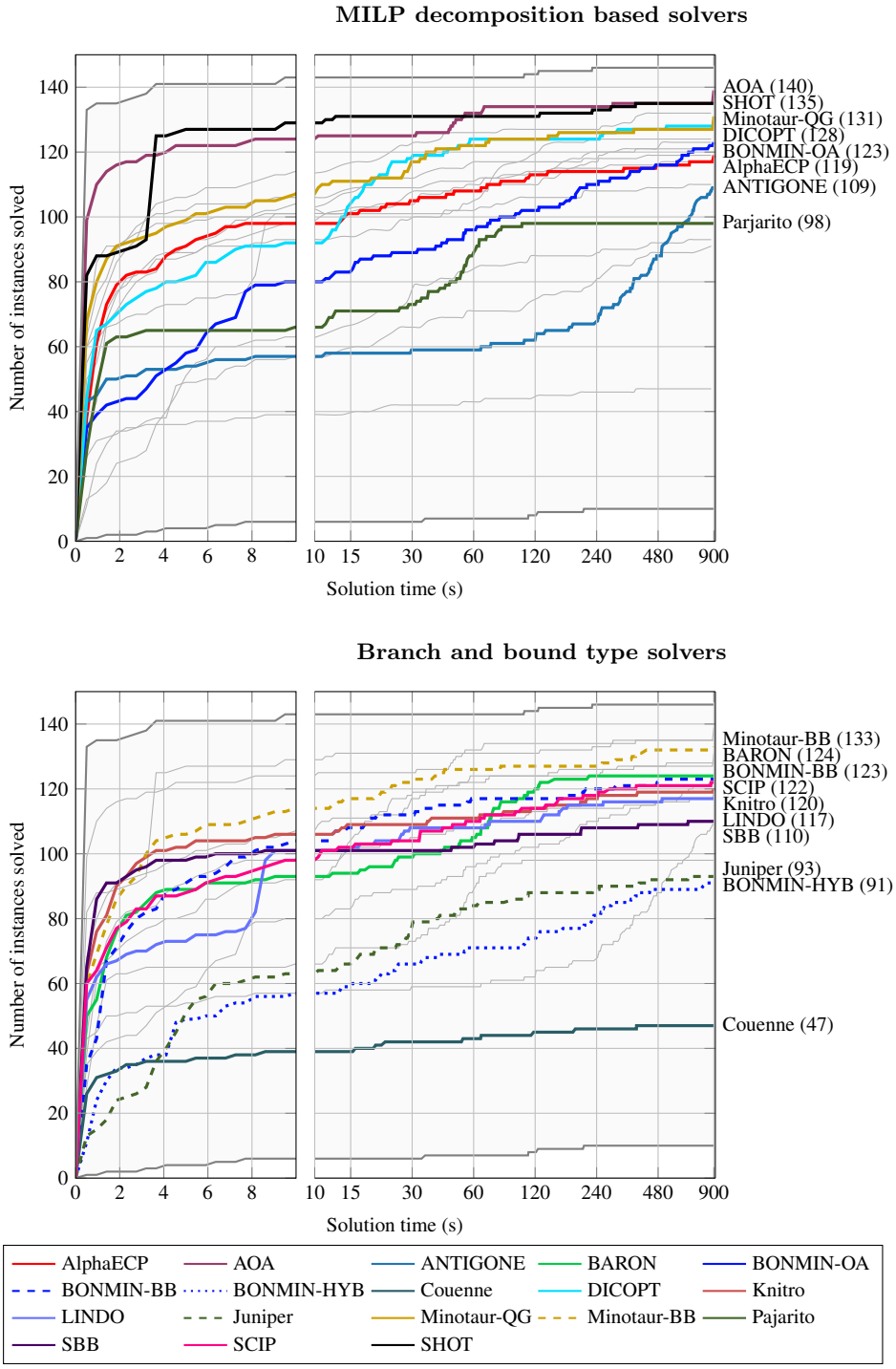

Figure 11: The solution profiles for problem instances with a high level of discrete variables as indicated in Appendix A.

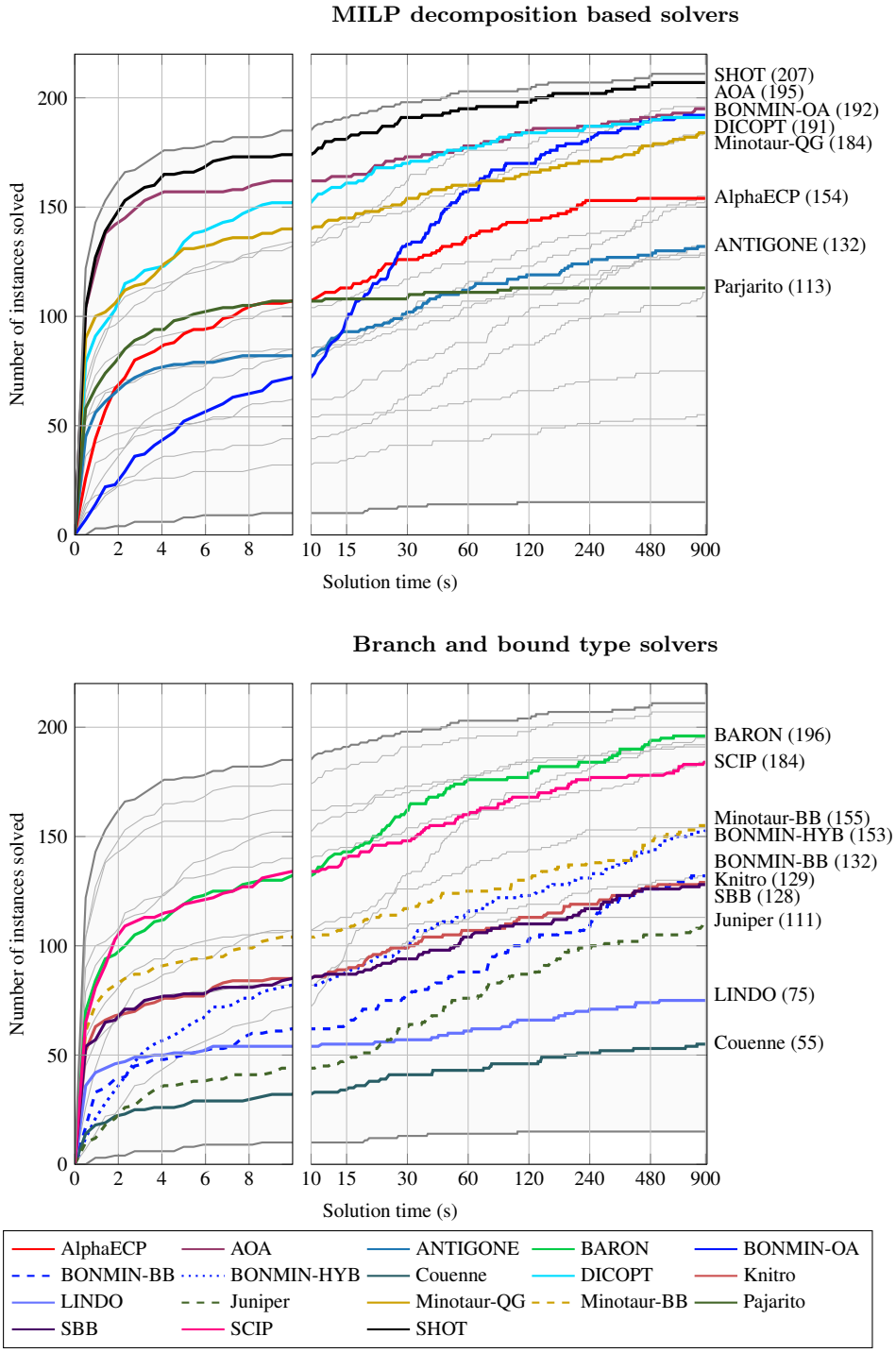

Figure 12: The solution profiles for problem instances with a low level of discrete variables as indicated in Appendix A.

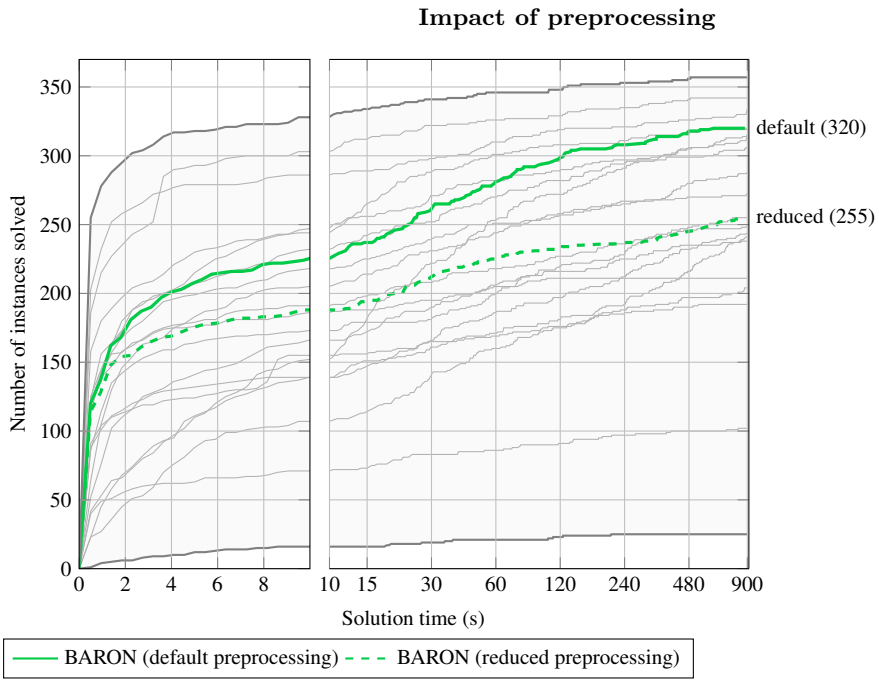

Figure 13: How preprocessing affect the performance of BARON.

Comparing the global solvers (ANTIGONE, BARON, and Couenne) with the convex solvers is not completely fair since the global solvers are able to solve a wider class of problems. Some of these solvers do not have a convex option, and thus, they have access to less information about the problem and might treat it as nonconvex. Other of the global solvers have a convex option which gives them this additional information, and some of the solvers, such as SHOT, simply assume that the problem is convex. Furthermore, we want to make a few comments on some of the solvers.

- In the comparison, Couenne does not perform as well as the other solvers. This is most likely due to failure in identifying the problems as convex and, therefore, it treats many of the problems as nonconvex and generates convex underestimators of the already convex constraints. In fact, Couenne is a nonconvex extension for the convex solver BONMIN, and there is really no reason to use it for convex problems. However, we included it in the comparison since it is a widely used general MINLP solver.

- The performance of ANTIGONE might also be affected by the solver not being able to identify the problems as convex. The solver might treat some of the convex functions as nonconvex, and therefore, generate unnecessarily weak relaxations.

- BARON seems to be very efficient at identifying the problems as convex since it is able to deal with the problems in such an efficient manner. Even if it is a global solver, capable of handling a variety of nonconvex problems, it is also one of the most efficient solvers for convex problems.

- Pajarito has mainly been developed to deal with MICP problems using a disciplined convex programming (DCP) approach, which is a different modeling paradigm based on a different problem formulation. The mixed-integer disciplinen convex programming (MIDCP) formulation enables Pajarito to utilize lifted problem formulations, resulting in tighter approximations. Here we have not used the MIDCP problem formulation, but the standard convex MINLP formulation. Reformulating the problems as MIDCP problems has been shown beneficial for Pajarito [92]. However, such formulations were not considered here and

Table 3: The table shows how the solvers are affected by the problem properties described in section 5.1. If a solver performs better for one of the categories it is indicated by a '+' sign, and a '–' sign indicates that the solver performs worse on that specific category. If the solver performs similarly on both categories it is indicated by '≈'. Furthermore, the number shows the total number of problems that the solver was able to solve within a relative objective gap of 0.1% within 900 seconds.

	Int. relaxation Gap		Nonlinearity		Discrete density	
MINLP solver	Hi.	Lo.	Hi.	Lo.	Hi.	Lo.
AlphaECP	≈ 98	≈ 175	≈ 101	≈ 172	+ 119	– 154
ANTIGONE	+ 110	– 131	≈ 88	≈ 153	≈ 109	≈ 132
AOA	≈ 132	≈ 202	≈ 126	≈ 209	≈ 140	≈ 195
BARON	≈ 134	≈ 186	– 110	+ 210	≈ 124	≈ 196
BONMINH-BB	– 57	+ 198	+ 117	– 138	+ 123	– 132
BONMINH-OA	≈ 122	≈ 193	≈ 107	≈ 208	≈ 123	≈ 192
BONMINH-HYB	– 79	+ 165	– 82	+ 162	– 91	+ 153
Couenne	≈ 36	≈ 66	≈ 40	≈ 62	≈ 47	≈ 55
DICOPT	≈ 121	≈ 198	- 106	+ 213	≈ 128	≈ 191
Juniper	– 50	– 154	+ 90	– 114	≈ 111	≈ 70
Knitro	– 57	+ 192	+ 112	– 137	+ 120	– 129
LINDO	– 36	+ 156	+ 105	– 87	+ 117	- 75
Minotaur-QG	≈ 120	≈ 195	≈ 112	≈ 203	≈ 131	≈ 184
Minotaur-BB	– 99	– 189	+ 119	– 169	+ 133	- 155
Pajarito	≈ 65?	≈ 111	– 51	+ 125	≈ 63?	≈ 113
SBB	– 52	+ 186	+ 110	– 128	+ 110	– 128
SCIP	≈ 124	≈ 181	- 95	+ 211	≈ 122	≈ 184
SHOT	≈ 139	≈ 203	≈ 121	≈ 221	≈ 135	≈ 207
Number of problems	151	215	133	233	151	215

at the moment the solver does not have to functionality to automatically reformulate the problems.

Based on the results presented here, one should not draw any conclusions on how the solvers perform on nonconvex problems. For example, SHOT which is the most efficient solver for this set of test problems does not, at the moment, have any functionality for dealing with nonconvex problems. Some of the other convex solvers may work quite well on nonconvex problems, of course without any guarantee of finding the global solution (or even a feasible solution).

7 Conclusions

The comparisons presented within this paper are mainly intended to help the readers make informed decisions about which tools to use when dealing with different types of convex MINLP problems. In the previous sections, we have shown how 15 different solvers performed on a test set containing 366 MINLP instances. By comparing the solvers on MINLP instances with different properties we noticed significant differences in the solvers' performance. For example, the solvers based on NLP-BB were strongly affected by both the integer relaxation gap and the degree of nonlinearity. Several of the solvers are based on the same main algorithms, although they differ significantly in terms of speed and number of problems solved. The differences are mainly due to different degrees of preprocessing, primal heuristics, cut generation procedures, and different strategies used by the solvers. The performance differences highlight the importance of such techniques for an efficient solver implementation.

For the test set considered here, SHOT and AOA were the overall fastest solvers. Both of

the solvers are based on a single tree approach similar to LP/NLP-BB and are closely integrated with the MILP solver by utilizing callbacks and adding the linearizations as lazy constraints. The results show the benefits of such a solution technique and support the strong belief in the single approach by [1] and [10]. The close integration with the MILP solver allows AOA and SHOT to benefit from the different techniques integrated within the MILP solver, such as branching heuristics, cut generation procedures, and bound tightening.

Overall several of the solvers performed well on the test set and were able to solve a large portion of the problems. The most instances any solver could solve within the time limit was 342 instances, and by combining all the solvers we where able solve 357 of the 366 MINLP problems to a 0.1% guaranteed optimality gap. However, it should be noted that many of the test instances are quite small and simple compared to industry-relevant problems. Still today, real-world problems must often be simplified and reduced in size to obtain tractable formulations, in the process limiting the practical benefits of MINLP. Thus, in order to fully benefit from convex MINLP as a tool for design and decision-making, both further algorithmic research and solver software development are required. We also hope that this paper encourages MINLP users to submit their instances to the instances libraries, *e.g.,* MINLPLib [100], which would benefit both MINLP solver developers and end users.

First, we want to thank the developers of the solvers for their support to the comparison, and we also want to thank both AIMMS (especially M. Hunting) and GAMS (especially S. Vigerske and M. Bussieck) for their support. D.E. Bernal and I.E. Grossmann would like to thank the Center for Advanced Process Decision-making (CAPD) for its financial support. Finally, A. Lundell wants to express his gratitude for the financial support from the Magnus Ehrnrooth Foundation, as well as the Ruth and Nils-Erik Stenbck Foundation.

References

[1] Abhishek K, Leyffer S, Linderoth J (2010) FilMINT: An outer approximation-based solver for convex mixed-integer nonlinear programs. INFORMS Journal on Computing 22(4):555–567

[2] Achterberg T (2009) SCIP: solving constraint integer programs. Mathematical Programming Computation 1(1):1–41

[3] Achterberg T, Wunderling R (2013) Mixed integer programming: Analyzing 12 years of progress. In: Facets of combinatorial optimization, Springer, pp 449–481

[4] Andersen ED, Andersen KD (2000) The MOSEK optimization software. EKA Consulting ApS, Denmark

[5] Balas E, Ceria S, Cornuéjols G (1993) A lift-and-project cutting plane algorithm for mixed 0–1 programs. Mathematical programming 58(1-3):295–324

[6] Bazaraa MS, Sherali HD, Shetty CM (2013) Nonlinear programming: theory and algorithms. John Wiley & Sons

[7] Belotti P (2010) Couenne: a user's manual. URL `https://www.coin-or.org/Couenne/couenne-user-manual.pdf`

[8] Belotti P, Lee J, Liberti L, Margot F, Wächter A (2009) Branching and bounds tightening techniques for non-convex MINLP. Optimization Methods and Software 24:597–634

[9] Belotti P, Cafieri S, Lee J, Liberti L (2010) Feasibility-based bounds tightening via fixed points. In: Wu W, Daescu O (eds) International Conference on Combinatorial Optimization and Applications, Springer, pp 65–76

[10] Belotti P, Kirches C, Leyffer S, Linderoth J, Luedtke J, Mahajan A (2013) Mixed-integer nonlinear optimization. Acta Numerica 22:1–131

[11] Benders JF (1962) Partitioning procedures for solving mixed-variables programming problems. Numerische mathematik 4(1):238–252

[12] Benson HY (2011) Mixed integer nonlinear programming using interior-point methods. Optimization Methods and Software 26(6):911–931

[13] Bernal DE, Vigerske S, Trespalacios F, Grossmann IE (2017) Improving the performance of dicopt in convex MINLP problems using a feasibility pump. Preprint, Optimization Online URL http://www.optimization-online.org/DB_HTML/2017/08/6171.html

[14] Bernal DE, Chen Q, Gong F, Grossmann IE (2018) Mixed-integer nonlinear decomposition toolbox for Pyomo (MindtPy). URL http://egon.cheme.cmu.edu/Papers/Bernal_Chen_MindtPy_PSE2018Paper.pdf

[15] Berthold T (2014) Heuristic algorithms in global MINLP solvers. PhD thesis, Technische Universitt Berlin

[16] Berthold T (2014) RENS — the optimal rounding. Mathematical Programming Computation 6(1):33–54

[17] Berthold T, Gleixner AM (2014) Undercover: a primal MINLP heuristic exploring a largest sub-MIP. Mathematical Programming 144(1-2):315–346

[18] Bezanson J, Karpinski S, Shah VB, Edelman A (2012) Julia: A fast dynamic language for technical computing. arXiv preprint:1209.5145

[19] Biegler LT (2010) Nonlinear programming: concepts, algorithms, and applications to chemical processes. SIAM

[20] Biegler LT, Grossmann IE (2004) Retrospective on optimization. Computers & Chemical Engineering 28(8):1169–1192

[21] Bisschop J (2006) AIMMS optimization modeling. Lulu.com

[22] Bonami P, Lee J (2007) BONMIN user's manual. Numer Math 4:1–32

[23] Bonami P, Lejeune MA (2009) An exact solution approach for portfolio optimization problems under stochastic and integer constraints. Operations Research 57(3):650–670

[24] Bonami P, Biegler LT, Conn AR, Cornuéjols G, Grossmann IE, Laird CD, Lee J, Lodi A, Margot F, Sawaya N, Wächter A (2008) An algorithmic framework for convex mixed integer nonlinear programs. Discrete Optimization 5(2):186–204

[25] Bonami P, Cornuéjols G, Lodi A, Margot F (2009) A feasibility pump for mixed integer nonlinear programs. Mathematical Programming 119(2):331–352

[26] Bonami P, Lee J, Leyffer S, Wächter A (2011) More branch-and-bound experiments in convex nonlinear integer programming. Preprint, Optimization online URL http://www.optimization-online.org/DB_FILE/2011/09/3191.pdf

[27] Bonami P, Kilinç M, Linderoth J (2012) Algorithms and software for convex mixed integer nonlinear programs. In: Mixed integer nonlinear programming, Springer, pp 1–39

[28] Borchers B, Mitchell JE (1994) An improved branch and bound algorithm for mixed integer nonlinear programs. Computers & Operations Research 21(4):359–367

[29] Boukouvala F, Misener R, Floudas CA (2016) Global optimization advances in mixed-integer nonlinear programming, MINLP, and constrained derivative-free optimization, CDFO. European Journal of Operational Research 252(3):701–727

[30] Bragalli C, DAmbrosio C, Lee J, Lodi A, Toth P (2012) On the optimal design of water distribution networks: a practical MINLP approach. Optimization and Engineering 13(2):219–246

[31] Brook A, Kendrick D, Meeraus A (1988) GAMS, a user's guide. ACM Signum Newsletter 23(3-4):10–11

[32] Bussieck MR, Vigerske S (2010) MINLP solver software. In: Wiley encyclopedia of operations research and management science, Wiley Online Library

[33] Bussieck MR, Dirkse SP, Vigerske S (2014) PAVER 2.0: an open source environment for automated performance analysis of benchmarking data. Journal of Global Optimization 59(2):259–275

[34] Byrd RH, Nocedal J, Waltz RA (2006) Knitro: An integrated package for nonlinear optimization. In: Large-scale nonlinear optimization, Springer, pp 35–59

[35] Cao W, Lim GJ (2011) Optimization models for cancer treatment planning. In: Wiley Encyclopedia of Operations Research and Management Science, Wiley Online Library

[36] Çezik MT, Iyengar G (2005) Cuts for mixed 0-1 conic programming. Mathematical Programming 104(1):179–202

[37] Currie J, Wilson DI, et al (2012) OPTI: lowering the barrier between open source optimizers and the industrial MATLAB user. Foundations of computer-aided process operations 24:32

[38] Dakin RJ (1965) A tree-search algorithm for mixed integer programming problems. The Computer Journal 8(3):250–255

[39] D'Ambrosio C, Frangioni A, Liberti L, Lodi A (2012) A storm of feasibility pumps for nonconvex MINLP. Mathematical programming 136(2):375–402

[40] Dolan ED, Moré JJ (2002) Benchmarking optimization software with performance profiles. Mathematical Programming, Series B 91(2):201–213

[41] Drud AS (1994) CONOPT — a large-scale GRG code. ORSA Journal on computing 6(2):207–216

[42] Dunning I, Huchette J, Lubin M (2017) JuMP: A modeling language for mathematical optimization. SIAM Review 59(2):295–320

[43] Duran MA, Grossmann IE (1986) An outer-approximation algorithm for a class of mixed-integer nonlinear programs. Mathematical Programming 36(3):307–339

[44] Eronen VP, Mäkelä MM, Westerlund T (2014) On the generalization of ecp and oa methods to nonsmooth convex MINLP problems. Optimization 63(7):1057–1073

[45] Eronen VP, Kronqvist J, Westerlund T, Mäkelä MM, Karmitsa N (2017) Method for solving generalized convex nonsmooth mixed-integer nonlinear programming problems. Journal of Global Optimization 69(2):443–459

[46] Exler O, Schittkowski K (2007) A trust region SQP algorithm for mixed-integer nonlinear programming. Optimization Letters 1(3):269–280

[47] FICO (2017) FICO Xpress-SLP manual. URL `https://www.artelys.com/uploads/pdfs/Xpress/Xpress_SLP_2795MS.pdf`

[48] FICO (2017) Xpress-optimizer reference manual. URL `https://www.artelys.com/uploads/pdfs/Xpress/Xpress_Optimizer_2447PS.pdf`

[49] Fischetti M, Lodi A (2011) Heuristics in mixed integer programming. In: Wiley Encyclopedia of Operations Research and Management Science, Wiley Online Library

[50] Fletcher R, Leyffer S (1994) Solving mixed integer nonlinear programs by outer approximation. Mathematical Programming 66(1):327–349

[51] Fletcher R, Leyffer S (1998) User manual for filterSQP. Numerical Analysis Report NA/181, Department of Mathematics, University of Dundee

[52] Floudas CA (1995) Nonlinear and mixed-integer optimization: fundamentals and applications. Oxford University Press

[53] Floudas CA (2000) Deterministic Global Optimization, vol. 37 of Nonconvex Optimization and its Applications

[54] Forrest J (2005) Cbc user's guide. URL `https://projects.coin-or.org/Cbc`

[55] Fourer R, Gay D, Kernighan B (1993) AMPL. Boyd & Fraser Danvers, MA

[56] Frangioni A, Gentile C (2006) Perspective cuts for a class of convex 0–1 mixed integer programs. Mathematical Programming 106(2):225–236

[57] GAMS (2018) Branch-and-Cut-and-Heuristic Facility. URL `https://www.gams.com/latest/docs/UG_SolverUsage.html`, [Accessed 18-May-2018]

[58] Geoffrion AM (1972) Generalized Benders decomposition. Journal of Optimization Theory and Applications 10(4):237–260

[59] Gill PE, Murray W, Saunders MA (2005) SNOPT: An SQP algorithm for large-scale constrained optimization. SIAM review 47(1):99–131

[60] Gleixner A, Eifler L, Gally T, Gamrath G, Gemander P, Gottwald RL, Hendel G, Hojny C, Koch T, Miltenberger M, Müller B, Pfetsch ME, Puchert C, Rehfeldt D, Schlösser F, Serrano F, Shinano Y, Viernickel JM, Vigerske S, Weninger D, Witt JT, Witzig J (2017) The SCIPOptimization Suite 5.0. Tech. Rep. 17-61, ZIB, Takustr. 7, 14195 Berlin

[61] Gomory RE, et al (1958) Outline of an algorithm for integer solutions to linear programs. Bulletin of the American Mathematical society 64(5):275–278

[62] Gould N, Scott J (2016) A note on performance profiles for benchmarking software. ACM Transactions on Mathematical Software (TOMS) 43(2):15

[63] Grossmann IE (1989) MINLP optimization strategies and algorithms for process synthesis. Tech. rep., Carnegie Mellon University

[64] Grossmann IE (2002) Review of nonlinear mixed-integer and disjunctive programming techniques. Optimization and engineering 3(3):227–252

[65] Grossmann IE, Kravanja Z (1997) Mixed-integer nonlinear programming: A survey of algorithms and applications. In: Biegler LT, Coleman TF, Conn AR, Santosa FN (eds) Large-scale optimization with applications, Springer, pp 73–100

[66] Gupta OK, Ravindran A (1985) Branch and bound experiments in convex nonlinear integer programming. Management science 31(12):1533–1546

[67] Gurobi (2018) Gurobi optimizer reference manual. Gurobi Optimization, LLC, URL `http://www.gurobi.com/documentation/8.0/refman.pdf`

[68] Hart WE, Laird CD, Watson JP, Woodruff DL, Hackebeil GA, Nicholson BL, Siirola JD (2012) Pyomo-optimization modeling in Python, vol 67. Springer

[69] Hijazi H, Bonami P, Ouorou A (2013) An outer-inner approximation for separable mixed-integer nonlinear programs. INFORMS Journal on Computing 26(1):31–44

[70] Holmström K (1999) The TOMLAB optimization environment in Matlab. AMO – Advanced Modeling and Optimization 1(1)

[71] HSL (2018) A collection of Fortran codes for large-scale scientific computation. http://www.hsl.rl.ac.uk

[72] Hunting M (2011) The AIMMS outer approximation algorithm for MINLP. Tech. rep., AIMMS B.V.

[73] IBM ILOG CPLEX Optimization Studio (2017) CPLEX Users Manual, version 12.7. IBM

[74] Inc LS (2017) LINDO User's Manual. URL https://www.lindo.com/downloads/PDF/LindoUsersManual.pdf

[75] Kelley JE Jr (1960) The cutting-plane method for solving convex programs. Journal of the Society for Industrial & Applied Mathematics 8(4):703–712

[76] Kılınç MR, Sahinidis NV (2018) Exploiting integrality in the global optimization of mixed-integer nonlinear programming problems with BARON. Optimization Methods and Software 33(3):540–562

[77] Kılınç MR, Linderoth J, Luedtke J (2017) Lift-and-project cuts for convex mixed integer nonlinear programs. Mathematical Programming Computation 9(4):499–526

[78] Kocis GR, Grossmann IE (1988) Global optimization of nonconvex mixed-integer nonlinear programming (MINLP) problems in process synthesis. Industrial & engineering chemistry research 27(8):1407–1421

[79] Kröger O, Coffrin C, Hijazi H, Nagarajan H (2018) Juniper: An Open-Source Nonlinear Branch-and-Bound Solver in Julia. arXiv preprint: 1804.07332

[80] Kronqvist J, Lundell A, Westerlund T (2016) The extended supporting hyperplane algorithm for convex mixed-integer nonlinear programming. Journal of Global Optimization 64(2):249–272

[81] Kronqvist J, Lundell A, Westerlund T (2017) A center-cut algorithm for solving convex mixed-integer nonlinear programming problems. In: Computer Aided Chemical Engineering, vol 40, Elsevier, pp 2131–2136

[82] Kronqvist J, Lundell A, Westerlund T (2018) Reformulations for utilizing separability when solving convex MINLP problems. Journal of Global Optimization pp 1–22

[83] Land AH, Doig AG (1960) An automatic method of solving discrete programming problems. Econometrica: Journal of the Econometric Society pp 497–520

[84] Lastusilta T (2002) GAMS MINLP solver comparisons and some improvements to the AlphaECP algorithm. PhD thesis, Åbo Akademi University

[85] Leyffer S (1993) Deterministic methods for mixed integer nonlinear programming. PhD University of Dundee

[86] Leyffer S (1999) User manual for MINLP BB. Tech. rep., University of Dundee numerical analysis report

[87] Leyffer S (2001) Integrating SQP and branch-and-bound for mixed integer nonlinear programming. Computational optimization and applications 18(3):295–309

[88] Liberti L (2009) Reformulation techniques in mathematical programming. HDR thesis

[89] Liberti L, Maculan N (2006) Global optimization: from theory to implementation, vol 84. Springer Science & Business Media

[90] Lin Y, Schrage L (2009) The global solver in the LINDO API. Optimization Methods & Software 24(4-5):657–668

[91] Lougee-Heimer R (2003) The common optimization interface for operations research: Promoting open-source software in the operations research community. IBM Journal of Research and Development 47(1):57–66

[92] Lubin M, Yamangil E, Bent R, Vielma JP (2016) Extended formulations in mixed-integer convex programming. In: Louveaux Q, Skutella M (eds) Integer Programming and Combinatorial Optimization: 18th International Conference, IPCO 2016, Springer International Publishing, pp 102–113

[93] Lundell A, Westerlund T (2017) Solving global optimization problems using reformulations and signomial transformations. Computers & Chemical Engineering (available online)

[94] Lundell A, Westerlund J, Westerlund T (2009) Some transformation techniques with applications in global optimization. Journal of Global Optimization 43(2-3):391–405

[95] Lundell A, Kronqvist J, Westerlund T (2017) SHOT – A global solver for convex MINLP in Wolfram Mathematica. In: Computer Aided Chemical Engineering, vol 40, Elsevier, pp 2137–2142

[96] Mahajan A, Leyffer S, Linderoth J, Luedtke J, Munson T (2017) Minotaur: A Mixed-Integer Nonlinear Optimization Toolkit. Preprint, Optimization Online URL `http://www.optimization-online.org/DB_FILE/2017/10/6275.pdf`

[97] Makhorin A (2008) GLPK (GNU linear programming kit). URL `http://www.gnu.org/software/glpk/`

[98] Melo W, Fampa M, Raupp F (2018) First steps to solve MINLP problems with Muriqui Optimizer. URL `http://www.wendelmelo.net/muriqui/manual_muriqui-ing3.pdf`, [Accessed 18-May-2018]

[99] Melo W, Fampa M, Raupp F (2018) An overview of MINLP algorithms and their implementation in Muriqui Optimizer. Annals of Operations Research pp 1–25

[100] MINLPLib (2018) Mixed-integer nonlinear programming library. URL `http://www.minlplib.org/`, [Accessed 27-May-2018]

[101] Misener R, Floudas CA (2009) Advances for the pooling problem: Modeling, global optimization, and computational studies. Applied and Computational Mathematics 8(1):3–22

[102] Misener R, Floudas CA (2014) ANTIGONE: Algorithms for continuous/integer global optimization of nonlinear equations. Journal of Global Optimization 59(2-3):503–526

[103] Nagarajan H, Lu M, Wang S, Bent R, Sundar K (2017) An adaptive, multivariate partitioning algorithm for global optimization of nonconvex programs. arXiv preprint: 1707.02514

[104] Nemhauser GL, Savelsbergh MW, Sigismondi GC (1994) MINTO, a MIxed INTeger optimizer. Operations Research Letters 15(1):47–58

[105] Nowak I, Alperin H, Vigerske S (2002) LaGO—an object oriented library for solving MINLPs. In: International Workshop on Global Optimization and Constraint Satisfaction, Springer, pp 32–42

[106] Nowak I, Breitfeld N, Hendrix EM, Njacheun-Njanzoua G (2018) Decomposition-based inner- and outer-refinement algorithms for global optimization. Journal of Global Optimization pp 1–17

[107] Palacios-Gomez F, Lasdon L, Engquist M (1982) Nonlinear optimization by successive linear programming. Management science 28(10):1106–1120

[108] Pörn R, Harjunkoski I, Westerlund T (1999) Convexification of different classes of non-convex MINLP problems. Computers & Chemical Engineering 23(3):439–448

[109] Quesada I, Grossmann IE (1992) An LP/NLP based branch and bound algorithm for convex MINLP optimization problems. Computers & chemical engineering 16(10-11):937–947

[110] Quist A, Van Geemert R, Hoogenboom J, Ílles T, Roos C, Terlaky T (1999) Application of nonlinear optimization to reactor core fuel reloading. Annals of Nuclear Energy 26(5):423–448

[111] Ryoo HS, Sahinidis NV (1996) A branch-and-reduce approach to global optimization. Journal of Global Optimization 8(2):107–138

[112] Sahinidis N, Grossmann IE (1991) MINLP model for cyclic multiproduct scheduling on continuous parallel lines. Computers & chemical engineering 15(2):85–103

[113] Sahinidis NV (2018) BARON user's manual. URL `https://minlp.com/downloads/docs/baron%20manual.pdf`

[114] Shectman JP, Sahinidis NV (1998) A finite algorithm for global minimization of separable concave programs. Journal of Global Optimization 12(1):1–36

[115] Stubbs RA, Mehrotra S (1999) A branch-and-cut method for 0-1 mixed convex programming. Mathematical programming 86(3):515–532

[116] Tawarmalani M, Sahinidis NV (2002) Convexification and global optimization in continuous and mixed-integer nonlinear programming: Theory, algorithms, software, and applications, vol 65. Springer Science & Business Media

[117] Tawarmalani M, Sahinidis NV (2005) A polyhedral branch-and-cut approach to global optimization. Mathematical Programming 103:225–249

[118] Trespalacios F, Grossmann IE (2014) Review of mixed-integer nonlinear and generalized disjunctive programming methods. Chemie Ingenieur Technik 86(7):991–1012

[119] Udell M, Mohan K, Zeng D, Hong J, Diamond S, Boyd S (2014) Convex optimization in Julia. SC14 Workshop on High Performance Technical Computing in Dynamic Languages `1410.4821`

[120] Vigerske S, Gleixner A (2018) SCIP: Global optimization of mixed-integer nonlinear programs in a branch-and-cut framework. Optimization Methods and Software 33(3):563–593

[121] Viswanathan J, Grossmann IE (1990) A combined penalty function and outer-approximation method for MINLPoptimization. Computers & Chemical Engineering 14(7):769–782

[122] Wächter A, Biegler LT (2006) On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Mathematical programming 106(1):25–57

[123] Westerlund T (2018) Users Guide for GAECP, An Interactive Solver for Generalized Convex MINLP-Problems using Cutting Plane and Supporting Hyperplane Techniques. URL `http://users.abo.fi/twesterl/GAECPDocumentation.pdf`

[124] Westerlund T, Petterson F (1995) An extended cutting plane method for solving convex MINLP problems. Computers & Chemical Engineering 19:131–136

[125] Westerlund T, Pörn R (2002) Solving pseudo-convex mixed integer optimization problems by cutting plane techniques. Optimization and Engineering 3(3):253–280

[126] Wolsey LA (1998) Integer Programming. Series in Discrete Mathematics and Optimization. Wiley-Interscience New Jersey

[127] Zhou K, Kılınç MR, Chen X, Sahinidis NV (2018) An efficient strategy for the activation of MIP relaxations in a multicore global MINLP solver. Journal of Global Optimization 70(3):497–516

[128] Zhu Y, Kuno T (2006) A disjunctive cutting-plane-based branch-and-cut algorithm for 0-1 mixed-integer convex nonlinear programs. Industrial & engineering chemistry research 45(1):187–196

Appendix A

A table of problem names and their classifications with regards to the different benchmark sets.

	relaxation		nonlinearity		discreteness	
instance name	gap	cat.	measure	cat.	measure	cat.
alan	0.89 %	LO	38 %	LO	50 %	LO
ball_mk2_10	–	HI	100 %	HI	100 %	HI
ball_mk2_30	–	HI	100 %	HI	100 %	HI
ball_mk3_10	–	HI	100 %	HI	100 %	HI
ball_mk3_20	–	HI	100 %	HI	100 %	HI
ball_mk3_30	–	HI	100 %	HI	100 %	HI
ball_mk4_05	–	HI	100 %	HI	100 %	HI
ball_mk4_10	–	HI	100 %	HI	100 %	HI
ball_mk4_15	–	HI	100 %	HI	100 %	HI
batch	9.2 %	LO	48 %	LO	52 %	HI
batch0812	5.6 %	LO	40 %	LO	60 %	HI
batchdes	3.9 %	LO	53 %	HI	47 %	LO
batchs101006m	4.5 %	LO	18 %	LO	46 %	LO
batchs121208m	3.1 %	LO	15 %	LO	50 %	LO
batchs151208m	2.8 %	LO	14 %	LO	46 %	LO
batchs201210m	1.7 %	LO	12 %	LO	45 %	LO
clay0203h	100 %	HI	20 %	LO	20 %	LO
clay0203m	100 %	HI	20 %	LO	60 %	HI
clay0204h	100 %	HI	15 %	LO	20 %	LO
clay0204m	100 %	HI	15 %	LO	62 %	HI
clay0205h	100 %	HI	12 %	LO	19 %	LO
clay0205m	100 %	HI	13 %	LO	63 %	HI
clay0303h	100 %	HI	27 %	LO	21 %	LO
clay0303m	100 %	HI	18 %	LO	64 %	HI
clay0304h	100 %	HI	20 %	LO	20 %	LO
clay0304m	100 %	HI	14 %	LO	64 %	HI
clay0305h	100 %	HI	16 %	LO	20 %	LO
clay0305m	100 %	HI	12 %	LO	65 %	HI
cvxnonsep_normcon20	0.34 %	LO	100 %	HI	50 %	LO
cvxnonsep_normcon20r	0.34 %	LO	50 %	LO	25 %	LO
cvxnonsep_normcon30	0.54 %	LO	100 %	HI	50 %	LO
cvxnonsep_normcon30r	0.54 %	LO	50 %	LO	25 %	LO
cvxnonsep_normcon40	0.78 %	LO	100 %	HI	50 %	LO
cvxnonsep_normcon40r	0.78 %	LO	50 %	LO	25 %	LO
cvxnonsep_nsig20	0.16 %	LO	100 %	HI	50 %	LO
cvxnonsep_nsig20r	0.16 %	LO	50 %	LO	25 %	LO
cvxnonsep_nsig30	0.11 %	LO	100 %	HI	50 %	LO
cvxnonsep_nsig30r	0.12 %	LO	50 %	LO	25 %	LO
cvxnonsep_nsig40	0.16 %	LO	100 %	HI	50 %	LO

instance name	relaxation gap	cat.	nonlinearity measure	cat.	discreteness measure	cat.
cvxnonsep_nsig40r	0.16 %	LO	50 %	LO	25 %	LO
cvxnonsep_pcon20	0.55 %	LO	100 %	HI	50 %	LO
cvxnonsep_pcon20r	0.55 %	LO	51 %	HI	26 %	LO
cvxnonsep_pcon30	0.47 %	LO	100 %	HI	50 %	LO
cvxnonsep_pcon30r	0.47 %	LO	51 %	HI	25 %	LO
cvxnonsep_pcon40	0.39 %	LO	100 %	HI	50 %	LO
cvxnonsep_pcon40r	0.39 %	LO	51 %	HI	25 %	LO
cvxnonsep_psig20	0.08 %	LO	100 %	HI	50 %	LO
cvxnonsep_psig20r	0.09 %	LO	50 %	LO	24 %	LO
cvxnonsep_psig30	0.32 %	LO	100 %	HI	50 %	LO
cvxnonsep_psig30r	0.32 %	LO	50 %	LO	24 %	LO
cvxnonsep_psig40	0.34 %	LO	100 %	HI	50 %	LO
cvxnonsep_psig40r	0.29 %	LO	50 %	LO	24 %	LO
du-opt	1.2 %	LO	100 %	HI	65 %	HI
du-opt5	51 %	HI	100 %	HI	65 %	HI
enpro48pb	6.9 %	LO	19 %	LO	60 %	HI
enpro56pb	15 %	LO	19 %	LO	57 %	HI
ex1223	15 %	LO	64 %	HI	36 %	LO
ex1223a	2.0 %	LO	43 %	LO	57 %	HI
ex1223b	15 %	LO	100 %	HI	57 %	HI
ex4	104 %	HI	14 %	LO	69 %	HI
fac1	0.11 %	LO	73 %	HI	27 %	LO
fac2	23 %	LO	82 %	HI	18 %	LO
fac3	30 %	LO	82 %	HI	18 %	LO
flay02h	25 %	LO	4.3 %	LO	8.7 %	LO
flay02m	25 %	LO	14 %	LO	29 %	LO
flay03h	37 %	LO	2.5 %	LO	10 %	LO
flay03m	37 %	LO	12 %	LO	46 %	LO
flay04h	43 %	LO	1.7 %	LO	10 %	LO
flay04m	43 %	LO	10 %	LO	57 %	HI
flay05h	46 %	LO	1.3 %	LO	10 %	LO
flay05m	46 %	LO	8.1 %	LO	65 %	HI
flay06h	48 %	LO	1.1 %	LO	11 %	LO
flay06m	48 %	LO	7.0 %	LO	70 %	HI
fo7	100 %	HI	12 %	LO	37 %	LO
fo7_2	100 %	HI	12 %	LO	37 %	LO
fo7_ar2_1	100 %	HI	13 %	LO	38 %	LO
fo7_ar25_1	100 %	HI	13 %	LO	38 %	LO
fo7_ar3_1	100 %	HI	13 %	LO	38 %	LO
fo7_ar4_1	100 %	HI	13 %	LO	38 %	LO
fo7_ar5_1	100 %	HI	13 %	LO	38 %	LO
fo8	100 %	HI	11 %	LO	38 %	LO
fo8_ar2_1	100 %	HI	11 %	LO	39 %	LO
fo8_ar25_1	100 %	HI	11 %	LO	39 %	LO
fo8_ar3_1	100 %	HI	11 %	LO	39 %	LO
fo8_ar4_1	100 %	HI	11 %	LO	39 %	LO
fo8_ar5_1	100 %	HI	11 %	LO	39 %	LO
fo9	100 %	HI	10 %	LO	40 %	LO
fo9_ar2_1	100 %	HI	10 %	LO	40 %	LO
fo9_ar25_1	100 %	HI	10 %	LO	40 %	LO
fo9_ar3_1	100 %	HI	10 %	LO	40 %	LO
fo9_ar4_1	100 %	HI	10 %	LO	40 %	LO
fo9_ar5_1	100 %	HI	10 %	LO	40 %	LO
gams01	97 %	HI	22 %	LO	76 %	HI
gbd	0.00 %	LO	25 %	LO	75 %	HI
hybriddynamic_fixed	10 %	LO	15 %	LO	14 %	LO

instance name	relaxation gap	cat.	nonlinearity measure	cat.	discreteness measure	cat.
ibs2	0.22 %	LO	100 %	HI	50 %	LO
jit1	0.37 %	LO	48 %	LO	16 %	LO
m3	100 %	HI	23 %	LO	23 %	LO
m6	100 %	HI	14 %	LO	35 %	LO
m7	100 %	HI	12 %	LO	37 %	LO
m7_ar2_1	100 %	HI	13 %	LO	38 %	LO
m7_ar25_1	100 %	HI	13 %	LO	38 %	LO
m7_ar3_1	100 %	HI	13 %	LO	38 %	LO
m7_ar4_1	100 %	HI	13 %	LO	38 %	LO
m7_ar5_1	100 %	HI	13 %	LO	38 %	LO
meanvarx	0.41 %	LO	20 %	LO	40 %	LO
meanvarxsc	0.41 %	LO	20 %	LO	67 %	HI
netmod_dol1	49 %	LO	0.3 %	LO	23 %	LO
netmod_dol2	16 %	LO	0.3 %	LO	23 %	LO
netmod_kar1	79 %	HI	0.9 %	LO	30 %	LO
netmod_kar2	79 %	HI	0.9 %	LO	30 %	LO
no7_ar2_1	100 %	HI	13 %	LO	38 %	LO
no7_ar25_1	100 %	HI	13 %	LO	38 %	LO
no7_ar3_1	100 %	HI	13 %	LO	38 %	LO
no7_ar4_1	100 %	HI	13 %	LO	38 %	LO
no7_ar5_1	100 %	HI	13 %	LO	38 %	LO
nvs03	49 %	LO	100 %	HI	100 %	HI
nvs10	0.74 %	LO	100 %	HI	100 %	HI
nvs11	0.41 %	LO	100 %	HI	100 %	HI
nvs12	0.41 %	LO	100 %	HI	100 %	HI
nvs15	89 %	HI	100 %	HI	100 %	HI
o7	100 %	HI	12 %	LO	37 %	LO
o7_2	100 %	HI	12 %	LO	37 %	LO
o7_ar2_1	100 %	HI	13 %	LO	38 %	LO
o7_ar25_1	100 %	HI	13 %	LO	38 %	LO
o7_ar3_1	100 %	HI	13 %	LO	38 %	LO
o7_ar4_1	100 %	HI	13 %	LO	38 %	LO
o7_ar5_1	100 %	HI	13 %	LO	38 %	LO
o8_ar4_1	100 %	HI	11 %	LO	39 %	LO
o9_ar4_1	100 %	HI	10 %	LO	40 %	LO
portfol_buyin	3.9 %	LO	47 %	LO	47 %	LO
portfol_card	4.0 %	LO	47 %	LO	47 %	LO
portfol_classical050_1	3.2 %	LO	33 %	LO	33 %	LO
portfol_classical200_2	13 %	LO	33 %	LO	33 %	LO
portfol_roundlot	0.00 %	LO	47 %	LO	47 %	LO
procurement2mot	37 %	LO	1.5 %	LO	7.5 %	LO
ravempb	15 %	LO	25 %	LO	48 %	LO
risk2bpb	1.0 %	LO	0.6 %	LO	3.0 %	LO
rsyn0805h	5.0 %	LO	2.9 %	LO	12 %	LO
rsyn0805m	63 %	HI	1.8 %	LO	41 %	LO
rsyn0805m02h	3.1 %	LO	2.6 %	LO	21 %	LO
rsyn0805m02m	160 %	HI	1.7 %	LO	41 %	LO
rsyn0805m03h	1.7 %	LO	2.6 %	LO	21 %	LO
rsyn0805m03m	104 %	HI	1.7 %	LO	41 %	LO
rsyn0805m04h	0.46 %	LO	2.6 %	LO	21 %	LO
rsyn0805m04m	57 %	HI	1.7 %	LO	41 %	LO
rsyn0810h	3.8 %	LO	5.2 %	LO	12 %	LO
rsyn0810m	72 %	HI	3.2 %	LO	40 %	LO
rsyn0810m02h	4.0 %	LO	4.6 %	LO	21 %	LO
rsyn0810m02m	298 %	HI	2.9 %	LO	41 %	LO
rsyn0810m03h	2.8 %	LO	4.6 %	LO	21 %	LO

instance name	relaxation gap	cat.	nonlinearity measure	cat.	discreteness measure	cat.
rsyn0810m03m	206 %	HI	2.9 %	LO	41 %	LO
rsyn0810m04h	0.93 %	LO	4.6 %	LO	21 %	LO
rsyn0810m04m	113 %	HI	2.9 %	LO	41 %	LO
rsyn0815h	6.7 %	LO	8.0 %	LO	12 %	LO
rsyn0815m	104 %	HI	5.4 %	LO	39 %	LO
rsyn0815m02h	4.2 %	LO	6.9 %	LO	21 %	LO
rsyn0815m02m	277 %	HI	4.7 %	LO	40 %	LO
rsyn0815m03h	3.1 %	LO	6.9 %	LO	21 %	LO
rsyn0815m03m	186 %	HI	4.7 %	LO	40 %	LO
rsyn0815m04h	1.7 %	LO	6.9 %	LO	21 %	LO
rsyn0815m04m	230 %	HI	4.7 %	LO	40 %	LO
rsyn0820h	7.3 %	LO	10 %	LO	12 %	LO
rsyn0820m	239 %	HI	6.5 %	LO	39 %	LO
rsyn0820m02h	1.2 %	LO	8.2 %	LO	21 %	LO
rsyn0820m02m	536 %	HI	5.5 %	LO	41 %	LO
rsyn0820m03h	3.6 %	LO	8.2 %	LO	21 %	LO
rsyn0820m03m	335 %	HI	5.5 %	LO	41 %	LO
rsyn0820m04h	2.4 %	LO	8.2 %	LO	21 %	LO
rsyn0820m04m	380 %	HI	5.5 %	LO	41 %	LO
rsyn0830h	10 %	LO	12 %	LO	13 %	LO
rsyn0830m	380 %	HI	8.0 %	LO	38 %	LO
rsyn0830m02h	6.1 %	LO	10 %	LO	21 %	LO
rsyn0830m02m	674 %	HI	6.5 %	LO	40 %	LO
rsyn0830m03h	3.0 %	LO	10 %	LO	21 %	LO
rsyn0830m03m	470 %	HI	6.5 %	LO	40 %	LO
rsyn0830m04h	2.0 %	LO	10 %	LO	21 %	LO
rsyn0830m04m	392 %	HI	6.5 %	LO	40 %	LO
rsyn0840h	8.2 %	LO	14 %	LO	13 %	LO
rsyn0840m	753 %	HI	10 %	LO	37 %	LO
rsyn0840m02h	5.8 %	LO	12 %	LO	21 %	LO
rsyn0840m02m	876 %	HI	7.8 %	LO	40 %	LO
rsyn0840m03h	2.3 %	LO	12 %	LO	21 %	LO
rsyn0840m03m	266 %	HI	7.8 %	LO	40 %	LO
rsyn0840m04h	2.1 %	LO	12 %	LO	21 %	LO
rsyn0840m04m	508 %	HI	7.8 %	LO	40 %	LO
slay04h	13 %	LO	5.7 %	LO	17 %	LO
slay04m	13 %	LO	18 %	LO	55 %	HI
slay05h	5.9 %	LO	4.3 %	LO	17 %	LO
slay05m	5.9 %	LO	14 %	LO	57 %	HI
slay06h	7.0 %	LO	3.5 %	LO	18 %	LO
slay06m	7.0 %	LO	12 %	LO	59 %	HI
slay07h	4.6 %	LO	2.9 %	LO	18 %	LO
slay07m	4.6 %	LO	10 %	LO	60 %	HI
slay08h	4.9 %	LO	2.5 %	LO	18 %	LO
slay08m	4.9 %	LO	8.7 %	LO	61 %	HI
slay09h	4.3 %	LO	2.2 %	LO	18 %	LO
slay09m	4.3 %	LO	7.7 %	LO	62 %	HI
slay10h	8.1 %	LO	2.0 %	LO	18 %	LO
slay10m	8.1 %	LO	6.9 %	LO	62 %	HI
smallinvDAXr1b010-011	1.8 %	LO	97 %	HI	97 %	HI
smallinvDAXr1b020-022	0.36 %	LO	97 %	HI	97 %	HI
smallinvDAXr1b050-055	0.10 %	LO	97 %	HI	97 %	HI
smallinvDAXr1b100-110	0.04 %	LO	97 %	HI	97 %	HI
smallinvDAXr1b150-165	0.03 %	LO	97 %	HI	97 %	HI
smallinvDAXr1b200-220	0.01 %	LO	97 %	HI	97 %	HI
smallinvDAXr2b010-011	1.8 %	LO	97 %	HI	97 %	HI

instance name	relaxation gap	cat.	nonlinearity measure	cat.	discreteness measure	cat.
smallinvDAXr2b020-022	0.36 %	LO	97 %	HI	97 %	HI
smallinvDAXr2b050-055	0.10 %	LO	97 %	HI	97 %	HI
smallinvDAXr2b100-110	0.04 %	LO	97 %	HI	97 %	HI
smallinvDAXr2b150-165	0.03 %	LO	97 %	HI	97 %	HI
smallinvDAXr2b200-220	0.01 %	LO	97 %	HI	97 %	HI
smallinvDAXr3b010-011	1.8 %	LO	97 %	HI	97 %	HI
smallinvDAXr3b020-022	0.36 %	LO	97 %	HI	97 %	HI
smallinvDAXr3b050-055	0.10 %	LO	97 %	HI	97 %	HI
smallinvDAXr3b100-110	0.04 %	LO	97 %	HI	97 %	HI
smallinvDAXr3b150-165	0.03 %	LO	97 %	HI	97 %	HI
smallinvDAXr3b200-220	0.01 %	LO	97 %	HI	97 %	HI
smallinvDAXr4b010-011	1.8 %	LO	97 %	HI	97 %	HI
smallinvDAXr4b020-022	0.36 %	LO	97 %	HI	97 %	HI
smallinvDAXr4b050-055	0.10 %	LO	97 %	HI	97 %	HI
smallinvDAXr4b100-110	0.04 %	LO	97 %	HI	97 %	HI
smallinvDAXr4b150-165	0.03 %	LO	97 %	HI	97 %	HI
smallinvDAXr4b200-220	0.01 %	LO	97 %	HI	97 %	HI
smallinvDAXr5b010-011	1.8 %	LO	97 %	HI	97 %	HI
smallinvDAXr5b020-022	0.36 %	LO	97 %	HI	97 %	HI
smallinvDAXr5b050-055	0.10 %	LO	97 %	HI	97 %	HI
smallinvDAXr5b100-110	0.04 %	LO	97 %	HI	97 %	HI
smallinvDAXr5b150-165	0.03 %	LO	97 %	HI	97 %	HI
smallinvDAXr5b200-220	0.01 %	LO	97 %	HI	97 %	HI
smallinvSNPr1b010-011	14 %	LO	99 %	HI	99 %	HI
smallinvSNPr1b020-022	7.9 %	LO	99 %	HI	99 %	HI
smallinvSNPr1b050-055	2.7 %	LO	99 %	HI	99 %	HI
smallinvSNPr1b100-110	1.2 %	LO	99 %	HI	99 %	HI
smallinvSNPr1b150-165	0.51 %	LO	99 %	HI	99 %	HI
smallinvSNPr1b200-220	0.53 %	LO	99 %	HI	99 %	HI
smallinvSNPr2b010-011	11 %	LO	99 %	HI	99 %	HI
smallinvSNPr2b020-022	10 %	LO	99 %	HI	99 %	HI
smallinvSNPr2b050-055	1.8 %	LO	99 %	HI	99 %	HI
smallinvSNPr2b100-110	1.3 %	LO	99 %	HI	99 %	HI
smallinvSNPr2b150-165	0.59 %	LO	99 %	HI	99 %	HI
smallinvSNPr2b200-220	0.59 %	LO	99 %	HI	99 %	HI
smallinvSNPr3b010-011	28 %	LO	99 %	HI	99 %	HI
smallinvSNPr3b020-022	7.5 %	LO	99 %	HI	99 %	HI
smallinvSNPr3b050-055	4.2 %	LO	99 %	HI	99 %	HI
smallinvSNPr3b100-110	1.8 %	LO	99 %	HI	99 %	HI
smallinvSNPr3b150-165	0.70 %	LO	99 %	HI	99 %	HI
smallinvSNPr3b200-220	0.49 %	LO	99 %	HI	99 %	HI
smallinvSNPr4b010-011	43 %	LO	99 %	HI	99 %	HI
smallinvSNPr4b020-022	11 %	LO	99 %	HI	99 %	HI
smallinvSNPr4b050-055	4.0 %	LO	99 %	HI	99 %	HI
smallinvSNPr4b100-110	1.0 %	LO	99 %	HI	99 %	HI
smallinvSNPr4b150-165	0.78 %	LO	99 %	HI	99 %	HI
smallinvSNPr4b200-220	0.22 %	LO	99 %	HI	99 %	HI
smallinvSNPr5b010-011	41 %	LO	99 %	HI	99 %	HI
smallinvSNPr5b020-022	15 %	LO	99 %	HI	99 %	HI
smallinvSNPr5b050-055	2.2 %	LO	99 %	HI	99 %	HI
smallinvSNPr5b100-110	1.4 %	LO	99 %	HI	99 %	HI
smallinvSNPr5b150-165	1.0 %	LO	99 %	HI	99 %	HI
smallinvSNPr5b200-220	0.31 %	LO	99 %	HI	99 %	HI
squfl010-025	51 %	HI	96 %	HI	3.8 %	LO
squfl010-040	43 %	LO	98 %	HI	2.4 %	LO
squfl010-080	49 %	LO	99 %	HI	1.2 %	LO

instance name	relaxation gap	cat.	nonlinearity measure	cat.	discreteness measure	cat.
squfl015-060	58 %	HI	98 %	HI	1.6 %	LO
squfl015-080	57 %	HI	99 %	HI	1.2 %	LO
squfl020-040	53 %	HI	98 %	HI	2.4 %	LO
squfl020-050	57 %	HI	98 %	HI	2.0 %	LO
squfl020-150	59 %	HI	99 %	HI	0.7 %	LO
squfl025-025	60 %	HI	96 %	HI	3.8 %	LO
squfl025-030	60 %	HI	97 %	HI	3.2 %	LO
squfl025-040	61 %	HI	98 %	HI	2.4 %	LO
squfl030-100	66 %	HI	99 %	HI	1.0 %	LO
squfl030-150	63 %	HI	99 %	HI	0.7 %	LO
squfl040-080	65 %	HI	99 %	HI	1.2 %	LO
sssd08-04	62 %	HI	6.7 %	LO	73 %	HI
sssd12-05	61 %	HI	5.3 %	LO	79 %	HI
sssd15-04	62 %	HI	4.5 %	LO	82 %	HI
sssd15-06	65 %	HI	4.5 %	LO	82 %	HI
sssd15-08	63 %	HI	4.5 %	LO	82 %	HI
sssd16-07	63 %	HI	4.3 %	LO	83 %	HI
sssd18-06	63 %	HI	4.0 %	LO	84 %	HI
sssd18-08	67 %	HI	4.0 %	LO	84 %	HI
sssd20-04	63 %	HI	3.7 %	LO	85 %	HI
sssd20-08	62 %	HI	3.7 %	LO	85 %	HI
sssd22-08	62 %	HI	3.4 %	LO	86 %	HI
sssd25-04	64 %	HI	3.1 %	LO	88 %	HI
sssd25-08	61 %	HI	3.1 %	LO	88 %	HI
st_e14	15 %	LO	64 %	HI	36 %	LO
st_miqp1	15 %	LO	100 %	HI	100 %	HI
st_miqp2	382 %	HI	50 %	LO	100 %	HI
st_miqp3	0.00 %	LO	50 %	LO	100 %	HI
st_miqp4	0.05 %	LO	50 %	LO	50 %	LO
st_miqp5	0.00 %	LO	29 %	LO	29 %	LO
st_test1	$3 \cdot 10^6$ %	HI	80 %	HI	100 %	HI
st_test2	9.5 %	LO	83 %	HI	100 %	HI
st_test3	24 %	LO	38 %	LO	100 %	HI
st_test4	11 %	LO	33 %	LO	100 %	HI
st_test5	104 %	HI	70 %	HI	100 %	HI
st_test6	30 %	LO	100 %	HI	100 %	HI
st_test8	0.00 %	LO	100 %	HI	100 %	HI
st_testgr1	0.11 %	LO	100 %	HI	100 %	HI
st_testgr3	0.63 %	LO	100 %	HI	100 %	HI
st_testph4	3.1 %	LO	100 %	HI	100 %	HI
stockcycle	1.7 %	LO	10 %	LO	90 %	HI
syn05h	0.03 %	LO	21 %	LO	12 %	LO
syn05m	37 %	LO	15 %	LO	25 %	LO
syn05m02h	0.02 %	LO	17 %	LO	19 %	LO
syn05m02m	19 %	LO	10 %	LO	33 %	LO
syn05m03h	0.02 %	LO	17 %	LO	19 %	LO
syn05m03m	20 %	LO	10 %	LO	33 %	LO
syn05m04h	0.01 %	LO	17 %	LO	19 %	LO
syn05m04m	20 %	LO	10 %	LO	33 %	LO
syn10h	0.03 %	LO	23 %	LO	13 %	LO
syn10m	58 %	HI	17 %	LO	29 %	LO
syn10m02h	0.08 %	LO	19 %	LO	21 %	LO
syn10m02m	104 %	HI	11 %	LO	36 %	LO
syn10m03h	0.05 %	LO	19 %	LO	21 %	LO
syn10m03m	103 %	HI	11 %	LO	36 %	LO
syn10m04h	0.04 %	LO	19 %	LO	21 %	LO

instance name	relaxation gap	cat.	nonlinearity measure	cat.	discreteness measure	cat.
syn10m04m	103 %	HI	11 %	LO	36 %	LO
syn15h	0.12 %	LO	26 %	LO	12 %	LO
syn15m	97 %	HI	20 %	LO	27 %	LO
syn15m02h	0.10 %	LO	21 %	LO	20 %	LO
syn15m02m	66 %	HI	13 %	LO	35 %	LO
syn15m03h	0.07 %	LO	21 %	LO	20 %	LO
syn15m03m	74 %	HI	13 %	LO	35 %	LO
syn15m04h	0.06 %	LO	21 %	LO	20 %	LO
syn15m04m	91 %	HI	13 %	LO	35 %	LO
syn20h	0.32 %	LO	26 %	LO	13 %	LO
syn20m	221 %	HI	22 %	LO	31 %	LO
syn20m02h	0.30 %	LO	21 %	LO	21 %	LO
syn20m02m	179 %	HI	13 %	LO	38 %	LO
syn20m03h	0.27 %	LO	21 %	LO	21 %	LO
syn20m03m	178 %	HI	13 %	LO	38 %	LO
syn20m04h	0.20 %	LO	21 %	LO	21 %	LO
syn20m04m	179 %	HI	13 %	LO	38 %	LO
syn30h	6.1 %	LO	25 %	LO	13 %	LO
syn30m	932 %	HI	20 %	LO	30 %	LO
syn30m02h	2.9 %	LO	20 %	LO	21 %	LO
syn30m02m	677 %	HI	13 %	LO	38 %	LO
syn30m03h	2.0 %	LO	20 %	LO	21 %	LO
syn30m03m	593 %	HI	13 %	LO	38 %	LO
syn30m04h	1.6 %	LO	20 %	LO	21 %	LO
syn30m04m	613 %	HI	13 %	LO	38 %	LO
syn40h	17 %	LO	26 %	LO	13 %	LO
syn40m	2608 %	HI	22 %	LO	31 %	LO
syn40m02h	2.4 %	LO	21 %	LO	21 %	LO
syn40m02m	1072 %	HI	13 %	LO	38 %	LO
syn40m03h	5.6 %	LO	21 %	LO	21 %	LO
syn40m03m	1467 %	HI	13 %	LO	38 %	LO
syn40m04h	2.0 %	LO	21 %	LO	21 %	LO
syn40m04m	917 %	HI	13 %	LO	38 %	LO
synthes1	87 %	HI	33 %	LO	50 %	LO
synthes2	101 %	HI	36 %	LO	45 %	LO
synthes3	78 %	HI	35 %	LO	47 %	LO
tls12	98 %	HI	19 %	LO	82 %	HI
tls2	86 %	HI	16 %	LO	89 %	HI
tls4	79 %	HI	19 %	LO	85 %	HI
tls5	89 %	HI	19 %	LO	84 %	HI
tls6	91 %	HI	20 %	LO	83 %	HI
tls7	96 %	HI	16 %	LO	86 %	HI
unitcommit1	1.6 %	LO	25 %	LO	75 %	HI
watercontamination0202	52 %	HI	3.8 %	LO	0.0 %	LO
watercontamination0202r	100 %	HI	48 %	LO	3.6 %	LO
watercontamination0303	64 %	HI	4.2 %	LO	0.0 %	LO
watercontamination0303r	100 %	HI	48 %	LO	3.6 %	LO

Appendix B

The options provided to the solvers (and subsolvers) in the benchmark are listed below. All other settings are the default values as provided by the individual solvers. Note that we have not tried to fine tune any of the solvers; however, if there is a convex strategy, or recommended convex parameters we have used those. We have also modified limits and other parameters when it is

apparent that implementation issues are the cause of, for example, premature termination of a solver on some problem instances. For example, without adding the CONOPT option specified below, AOA, Dicopt and SBB fails to solve all the `smallinv`-instances in MINLPLib. Therefore, we believe that it is motivated to use these, since this problem occurs in the subsolver.

name	value	description
General GAMS		
MIP	CPLEX	uses CPLEX as MIP solver
threads	8	max amount of threads
optcr	0.001	relative termination tolerance
optca	0.001	absolute termination tolerance
nodlim	10^8	to avoid premature termination
domlim	10^8	to avoid premature termination
iterlim	10^8	to avoid premature termination
reslim	900	time limit
AlphaECP		
ECPmaster	1	activates convex strategy
AOA		
IsConvex	1	activates convex strategy
IterationMax	10^7	maximal number of iterations
RelativeOptimalityTolerance	0.1	relative termination tolerance
TimeLimit	900	time limit
BONMIN		
bonmin.algorithm	B-OA B-HYB B-BB	selects the main algorithm
milp_solver	CPLEX	uses CPLEX as MILP solver
bonmin.time_limit	900	sets the time limit
Couenne		
lp_solver	CPLEX	uses CPLEX as LP solver
DICOPT		
convex	1	activates convex strategy
stop	1	convex stopping criterion
maxcycles	10^8	iteration limit
infeasder	1	add cutting planes from infeasible NLP (convex recommendation)
nlpoptfile	1	to use the CONOPT options below
Juniper		
fp_cpx		use CPLEX for the feasibility pump
processors	8	max number of threads
time-limit	900	time limit
LINDO		
usegop	0	deactivates global strategy
splex_itrlmt	-1	simplex iteration limit
MIP_itrlim	-1	MILP iteration limit iteration limits are set as infinite to avoid premature termination
Minotaur		

bnb_time_limit	900	time limit
lp_engine	OsiCpx	use CPLEX as MIP solver
obj_gap_percent	10^{-5}	relative termination tolerance
threads	8	max amount of threads
Pajarito		
cpx		use CPLEX as MILP solver
processors	8	max number of threads
time-limit	900	time limit
SBB		
memnodes	$5 \cdot 10^7$	to avoid premature termination, but not too large, since memory is preallocated
rootsolver	CONOPT.1	to use the CONOPT options below
SCIP		
constraints/nonlinear/ assumeconvex	true	activates convex strategy
SHOT		
Dual.MIP.NumberOfThreads	8	max number of threads
Dual.MIP.Solver	0	use CPLEX as MIP solver
Primal.FixedInteger.Solver	2	to use GAMS NLP solvers
Subsolver.GAMS.NLP.Solver	conopt	use CONOPT as GAMS NLP solver
Termination.ObjectiveGap. Absolute	0.001	absolute termination tolerance
Termination.ObjectiveGap. Relative	0.001	relative termination tolerance
Termination.TimeLimit	900	time limit
CONOPT (AIMMS)		
Maximal solution of a variable	10^{29}	to avoid problems with unbounded variables in AOA; 10^{29} is the max value
CONOPT (GAMS)		
rtmaxv	10^{30}	to avoid problems with unbounded variables in DICOPT and SBB