CrossMark

An adaptive, multivariate partitioning algorithm for global optimization of nonconvex programs

Harsha Nagarajan[1] · Mowen Lu[2] · Site Wang[2] · Russell Bent[1] · Kaarthik Sundar[1]

Abstract

In this work, we develop an adaptive, multivariate partitioning algorithm for solving nonconvex, Mixed-Integer Nonlinear Programs (MINLPs) with polynomial functions to global optimality. In particular, we present an iterative algorithm that exploits piecewise, convex relaxation approaches via disjunctive formulations to solve MINLPs that is different than conventional spatial branch-and-bound approaches. The algorithm partitions the domains of variables in an adaptive and non-uniform manner at every iteration to focus on productive areas of the search space. Furthermore, domain reduction techniques based on sequential, optimization-based bound-tightening and piecewise relaxation techniques, as a part of a presolve step, are integrated into the main algorithm. Finally, we demonstrate the effectiveness of the algorithm on well-known benchmark problems (including Pooling and Blending instances) from MINLPLib and compare our algorithm with state-of-the-art global optimization solvers. With our novel approach, we solve several large-scale instances, some of which are not solvable by state-of-the-art solvers. We also succeed in reducing the best known optimality gap for a hard, generalized pooling problem instance.

Keywords Global optimization · Adaptive partitioning · MILP-based methods · Mixed-integer nonlinear programs · McCormick · Piecewise relaxations · Sequential optimization-based bound-tightening

✉ Harsha Nagarajan
  harsha@lanl.gov

  Mowen Lu
  mlu87@g.clemson.edu

  Site Wang
  sitew@g.clemson.edu

  Russell Bent
  rbent@lanl.gov

  Kaarthik Sundar
  kaarthik@lanl.gov

1   Center for Nonlinear Studies, Los Alamos National Laboratory, New Mexico, USA

2   Department of Industrial Engineering, Clemson University, Clemson, USA

🖄 Springer

1 Introduction

Mixed-Integer Nonlinear Programs (MINLPs) are convex/non-convex, mathematical programs that include discrete variables and nonlinear terms in the objective function and/or constraints. In practice, non-convex MINLPs arise in many applications such as chemical engineering (synthesis of process and water networks) [33,45], energy infrastructure networks [20,30,41], and in molecular distance geometry problems [28], to name a few. Given the importance of these problems, considerable research has been devoted to developing approaches for solving MINLPs, such as approaches implemented in such solvers as BARON [47], Couenne [5] and SCIP [1]. Within these approaches, two of the key features of successful methods include MINLP relaxations and search. For example, in a typical solver, non-convex terms are replaced with convex over- and under-estimators [8]. The resulting convex optimization problem is a relaxation of the original MINLP and its solution is a bound to the optimal objective value of the MINLP. These relaxations are then used in conjunction with a search procedure, like spatial branch-and-bound (sBB), to explore the solution space of the MINLP and identify the global optimal solution.

Despite major developments related to these features and others, MINLPs still remain difficult to solve and global optimization solvers often struggle to find optimal solutions and at times, even a feasible solution. In many cases, the source of these struggles are weak relaxations of the MINLP and the impact weak relaxations on the size of the search space that is explored. To address these difficulties, in this paper we develop an approach for deriving better bounds through piecewise convex relaxations that are modeled as mixed-integer convex optimization problems. The piecewise convex relaxations are combined with additional algorithmic enhancements, a novel, adaptive domain partitioning scheme, and successive solves of mixed-integer problems (MIP), to produce a novel search procedure. This global optimization algorithm is tested extensively on MINLPs with polynomial constraints, including the well-known and hard Pooling and Blending instances [26] and is compared with state-of-the-art global optimization approaches. In this paper, we focus on MINLPs with polynomial constraints, but the approach is fairly generic and can be generalized to other nonconvex functions. Finally, for ease of exposition, we assume the MINLPs are minimization problems throughout the rest of the paper.

We next discuss the key contributions we make in this paper. Our first contribution improves convex relaxations of polynomial functions. Here, we develop an approach based on piecewise convex relaxations. While such relaxations have been used to bound medium-sized MINLPs with bilinear functions [7,12,19,24,26], we generalize these approaches to arbitrary polynomial functions.

Our second contribution turns the derivation of these relaxations into a search procedure based on solving MIPs, i.e., a 'MIP-based approach' that is akin to the approach of [17,44]. Most existing approaches rely on sBB. In a conventional sBB algorithm, branching occurs on the domain of one variable at a time. The branching generates two new problems (child nodes), each with a smaller domain than the parent problem (node) and potentially tighter relaxations. Whenever the best possible solution at a node is worse than the best known feasible solution, the node is pruned. sBB is typically combined with with enhancements such as cutting planes and domain reduction techniques to further improve the efficiency of the search [47,49]. In contrast, our MIP-based approach solves a sequence of MIPs based on successively tighter piecewise convex relaxations that converges to the optimal solution.

Our third contribution is a sparse domain partitioning approach for piecewise convex relaxations. Most existing partitioning approaches rely on uniform partitioning, i.e., [19].

Unfortunately, uniform partitioning, when used in conjunction with a MIP-based approach can lead to MIPs with a large number of binary variables. Thus, uniform partitioning limits MIP based approaches to small- and medium-sized problems. This important issue has motivated the development of piecewise relaxation techniques where the number of binary variables increases logarithmically [36,52] with the number of partitions and multiparametric disaggregation approaches [11]. In other work [53], the authors present a non-uniform, bivariate partitioning approach that improves the relaxations but provide results for a single, simple benchmark problem. Reference [50] discusses a univariate parametrization method applied to medium-sized benchmarks. However, none of these approaches address the key limitation of uniform partitioning, *partition density*, i.e. these methods introduce partitions in unproductive areas of the variable domains. We address this limitation by introducing a novel approach that adaptively partitions the relaxations in regions of the search space that favor optimality. To the best of our knowledge, this is the first work in the literature that develops a complete MIP-based method for solving MINLPs to global optimality based on sparse domain partitioning schemes.

Our fourth (minor) contribution combines the adaptively partitioned piecewise relaxation approach with sequential, optimization-based bound-tightening (OBBT). OBBT is used as a presolve step in the overall global optimization algorithm. OBBT solves a sequence of convex minimization and maximization problems on the variables that appear in nonconvex terms. The solutions to these problems tighten domains of the variables and the associated relaxation to the nonconvex terms [4,16,37,42]. Recent work has observed the effectiveness of applying OBBT in various applications [13,39,54]. We adapt and extend this approach by solving *convex MIPs* in the OBBT procedure (existing approaches solve ordinary convex problems). Though this approach seems counter-intuitive, computational experiments indicate that the value of the strengthened bounds obtained by solving MIPs often outweigh the computational time required to solve them.

Finally, these four contributions are combined into a MIP-based global optimization algorithm which is referred to as the Adaptive, Multivariate Partitioning (AMP) algorithm. Given an MINLP, AMP first calculates a local solution to the MINLP, an initial lower bound, and tightened variable bounds (sequential OBBT) as a presolve step. The main loop of the AMP algorithm refines the partitions of the variable domain, computes improved lower bounds, and derives better local (upper bound) solutions. The variable domains are refined in a non-uniform and adaptive fashion. In particular, partitions are dynamically added around the optimal solution to the relaxed problem at each iteration of AMP. This loop iterates until the relative gap between the lower bound and the upper bound solution meets a user specified global optimality tolerance. The computation may also be interrupted early to provide a local optimal solution.

A preliminary version of this work [38] was applied to hard, infrastructure network optimization problems [29,54], which demonstrated the effectiveness of adaptive partitioning strategies. Given the efficacy of the proposed ideas, including various enhancements (not discussed in this paper), the algorithm's implementation is also available as an open-source solver in Julia programming language [6]. The remainder of this paper is organized as follows: Sect. 2 discusses the required notation, problem set-up, and reviews standard convex relaxations. Section 3 discusses our Adaptive, Multivariate Partitioning Algorithm to solve MINLPs to global optimality with a few proofs of convergence guarantees. Section 4 illustrates the strength of the algorithms on benchmark MINLPs and Sect. 5 concludes the paper.

2 Definitions

Notation Here, we use lower and upper case for vector and matrix entries, respectively. Bold font refers to the entire vector or matrix. With this notation, $||v||_\infty$ defines the ℓ^∞ norm of vector $v \in \mathbb{R}^n$. Given vectors $v_1 \in \mathbb{R}^n$ and $v_2 \in \mathbb{R}^n$, $v_1 \cdot v_2 = \sum_{i=1}^n v_{1i} v_{2i}$; $v_1 + v_2$ implies element-wise sums; and $\frac{v_1}{\alpha}$ denotes the element-wise ratio between entries of v_1 and the scalar α. Next, $z \in \mathbb{Z}$ represents an integer (variable/constant) and specifically $z \in \mathbb{B}$ represents a binary variable. Finally, we let e_i denote a unit vector whose ith coordinate is one.

Problem The problems considered in this paper are MINLPs with polynomials which have at least one feasible solution. The general form of the problem, denoted as $\mathcal{P}$, is as follows:

$$\mathcal{P}: \underset{x,y}{\text{minimize}} \quad f(x, y)$$
$$\text{subject to} \quad g(x, y) \leqslant 0,$$
$$h(x, y) = 0,$$
$$x^L \leqslant x \leqslant x^U,$$
$$y \in \{0, 1\}^m$$

where, $f : \mathbb{R}^n \times \mathbb{B}^m \to \mathbb{R}$, $g_i : \mathbb{R}^n \times \mathbb{B}^m \to \mathbb{R}$ for $i = 1, \dots, G$ and $h_i : \mathbb{R}^n \times \mathbb{B}^m \to \mathbb{R}$ for $i = 1, \dots, H$ are polynomials. For the sake of clarity, neglecting the binary variables in the functions, f, g or h can assume the following form:

$$\sum_{t \in T} a_t \prod_{k \in K_t} x_k^{\alpha_k} \tag{1}$$

where, T is a set of terms in a polynomial, K_t is a set of variables in term t, $a_t \in \mathbb{R}$ is a real coefficient and α_k is an exponent (integer) value. x and y are vectors of continuous variables with box constraints $[x^L, x^U]$ and binary variables, respectively. x and y have dimension n and m, respectively. We use notation σ to denote a solution to $\mathcal{P}$, where $\sigma(\cdot)$ is the value of variable(s), $\cdot$, in σ and $f(\sigma)$ is the objective value of σ. We note that $\mathcal{P}$ is an NP-hard combinatorial problem. The construction of convex relaxations for each individual term in Eq. (1) plays a critical role in developing algorithms for solving $\mathcal{P}$ to global optimality. In the following paragraphs, we discuss the relaxations used in this paper.

In this paper, we use relaxations for bilinear, multilinear and quadratic monomials. Note that, without loss of generality, any polynomial can be equivalently expressed using a combination of these monomials.

McCormick relaxation of a bilinear term For $t \in T$, when $|K_t| \leqslant 2$ and $\alpha_k = 1$, the McCormick relaxation [32] is used. Given variables x_i and x_j that appear in t, McCormick relaxed the set

$$S_B = \left\{ (x_i, x_j, \widehat{x_{ij}}) \in [x_i^L, x_i^U] \times [x_j^L, x_j^U] \times \mathbb{R} \mid \widehat{x_{ij}} = x_i x_j \right\}$$

with the following four inequalities:

$$\widehat{x_{ij}} \geqslant x_i^L x_j + x_j^L x_i - x_i^L x_j^L \tag{2a}$$
$$\widehat{x_{ij}} \geqslant x_i^U x_j + x_j^U x_i - x_i^U x_j^U \tag{2b}$$
$$\widehat{x_{ij}} \leqslant x_i^L x_j + x_j^U x_i - x_i^L x_j^U \tag{2c}$$
$$\widehat{x_{ij}} \leqslant x_i^U x_j + x_j^L x_i - x_i^U x_j^L \tag{2d}$$

Let $\langle x_i, x_j \rangle^{MC} \supset S_B$ represent the feasible region defined by (2). For a single bilinear term $x_i x_j$, the relaxations in (2) describe the convex hull of set S_B [2].

Recursive McCormick relaxation of a multilinear term For a general multilinear term ($|K_t| \geqslant 3$, $\alpha_k = 1$), McCormick proposed a recursive approach to successively derive envelopes on bilinear combinations of the terms. The resulting relaxation has formed the basis for the relaxations used in the global optimization literature, including the implementations in BARON, Couenne and SCIP [1,5,47]. More formally, the non-convex function given by $\prod_{k=1}^{|K_t|} x_k$ can be relaxed by introducing lifted variables $\widehat{x}_1, \ldots, \widehat{x}_{|K_t|-1}$ such that $\widehat{x}_1 = x_1 x_2$ and $\widehat{x}_i = \widehat{x}_{i-1} x_{i+1}$ for every $i = 2, \ldots, |K_t| - 1$. Thus, the recursive McCormick envelopes of $\prod_{k=1}^{|K_t|} x_k$ are described by

$$\left\{ (x_1, x_2, \widehat{x}_1) \in [x_1^L, x_1^U] \times [x_2^L, x_2^U] \times [\widehat{x}_1^L, \widehat{x}_1^U] \mid \widehat{x}_1 = \langle x_1, x_2 \rangle^{MC} \right\}, \tag{3a}$$

$$\left\{ (\widehat{x}_{i-1}, x_{i+1}, \widehat{x}_i) \in [\widehat{x}_{i-1}^L, \widehat{x}_{i-1}^U] \times [x_{i+1}^L, x_{i+1}^U] \times [\widehat{x}_i^L, \widehat{x}_i^U] \mid \right.$$
$$\left. \widehat{x}_i = \langle \widehat{x}_{i-1}, x_{i+1} \rangle^{MC} \right\}, \quad \forall i = 2, \ldots, |K_t| - 1. \tag{3b}$$

where, the bounds of $\widehat{x}_i$ variables are derived appropriately. By abuse of notation, (3) can be succinctly represented as

$$\left\langle \prod_{k=1}^{|K_t|} x_k \right\rangle^{MC} = \left\langle \left\langle \langle x_1, x_2 \rangle^{MC}, \ldots, x_{|K_t|-1} \right\rangle^{MC}, x_{|K_t|} \right\rangle^{MC}.$$

In general, the recursive McCormick envelopes described in (3) for a single multilinear term are not the tightest possible relaxation. The choice of the recursion order affects the tightness of the relaxation [10,48]. However, authors in [46] prove that (3) describes the convex hull when the bounds on the variables are in the set [0, 1]. This result was generalized by [31] for variables with bounds that are either $[0, x_i^U]$ or $[-x_i^U, x_i^U]$ (symmetric about the origin). More generally, the convex hull of a multilinear term can be obtained by using an extreme point characterization by using exponential number of variables [43]. The computational tractability of using an extreme point characterization for piecewise relaxation of multilinear terms remains a subject of future work.

Piecewise McCormick relaxation of a bilinear term In the presence of partitions on the variables involved in a multilinear term, the McCormick relaxations (applied on bilinear terms) can be tightened (see Fig. 1a) by using a piecewise convex relaxation which uses one binary variable per variable partition. Given a bilinear term $x_i x_j$ and partition sets $\mathcal{I}_i$ and $\mathcal{I}_j$, binary variables $\widehat{y}_i \in \{0, 1\}^{|\mathcal{I}_i|}$ and $\widehat{y}_j \in \{0, 1\}^{|\mathcal{I}_j|}$ are used to denote these partitions. Each entry blue in $\mathcal{I}_i$ is a pair of values, $\langle i, j \rangle$ that model the upper and lower bound of a variable in a partition. We refer to the collection of all partition sets with $\mathcal{I}$. These binary variables are used to control the partitions that are active and the associated relaxation of the active partition. Formally, the piecewise McCormick constraints, denoted by $\widehat{x_{ij}} \in \langle x_i, x_j \rangle^{MC(\mathcal{I})}$, take the following form:

$$\widehat{x_{ij}} \geqslant \left(x_i^l \cdot \widehat{y}_i \right) x_j + \left(x_j^l \cdot \widehat{y}_j \right) x_i - \left(x_i^l \cdot \widehat{y}_i \right) \left(x_j^l \cdot \widehat{y}_j \right) \tag{4a}$$

$$\widehat{x_{ij}} \geqslant \left(x_i^u \cdot \widehat{y}_i \right) x_j + \left(x_j^u \cdot \widehat{y}_j \right) x_i - \left(x_i^u \cdot \widehat{y}_i \right) \left(x_j^u \cdot \widehat{y}_j \right) \tag{4b}$$

$$\widehat{x_{ij}} \leqslant \left(x_i^l \cdot \widehat{y}_i \right) x_j + \left(x_j^u \cdot \widehat{y}_j \right) x_i - \left(x_i^l \cdot \widehat{y}_i \right) \left(x_j^u \cdot \widehat{y}_j \right) \tag{4c}$$
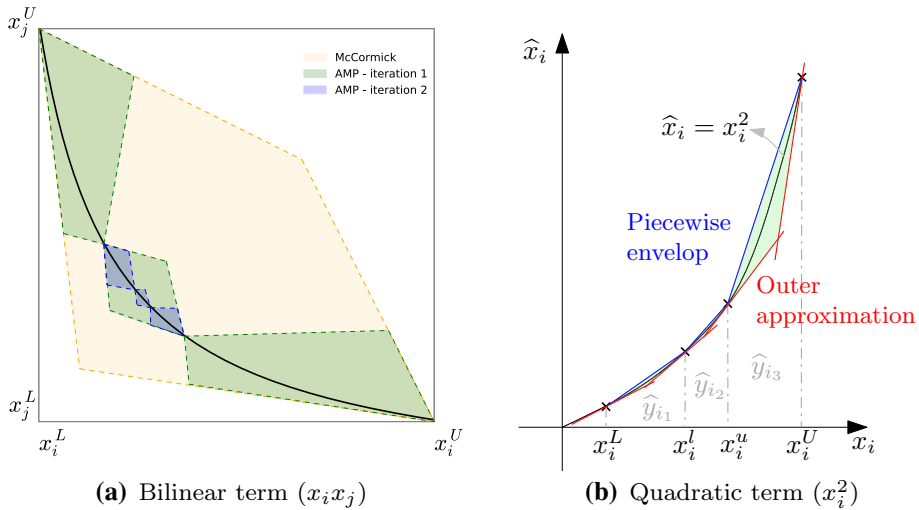
(a) Bilinear term $(x_i x_j)$

(b) Quadratic term (x_i^2)

Fig. 1 Piecewise relaxations (shaded) of bilinear and quadratic terms for a given set of partitions

$$\widehat{x_{ij}} \leqslant \left(\boldsymbol{x}_i^u \cdot \widehat{\boldsymbol{y}}_i\right) x_j + \left(\boldsymbol{x}_j^l \cdot \widehat{\boldsymbol{y}}_j\right) x_i - \left(\boldsymbol{x}_i^u \cdot \widehat{\boldsymbol{y}}_i\right) \left(\boldsymbol{x}_j^l \cdot \widehat{\boldsymbol{y}}_j\right) \tag{4d}$$

$$\widehat{\boldsymbol{y}}_i \cdot \mathbf{1} = 1, \ \widehat{\boldsymbol{y}}_j \cdot \mathbf{1} = 1 \tag{4e}$$

$$\widehat{\boldsymbol{y}}_i \in \{0, 1\}^{|\mathcal{I}_i|}, \quad \widehat{\boldsymbol{y}}_j \in \{0, 1\}^{|\mathcal{I}_j|} \tag{4f}$$

where, $(\boldsymbol{x}_i^l, \boldsymbol{x}_i^u) \in \mathcal{I}_i$ are the vector form of the partition sets of x_i $(\mathcal{I}_i)$ and $\mathbf{1}$ is a vector of ones of appropriate dimension. Also, $(\boldsymbol{x}_i^l \cdot \widehat{\boldsymbol{y}}_i)(\boldsymbol{x}_j^l \cdot \widehat{\boldsymbol{y}}_j)$ is rewritten as $\boldsymbol{x}_i^l(\widehat{\boldsymbol{y}}_i \widehat{\boldsymbol{y}}_j^T)\boldsymbol{x}_j^l$, where $(\widehat{\boldsymbol{y}}_i \widehat{\boldsymbol{y}}_j^T)$ is a matrix with binary product entries. Note that these binary products and the bilinear terms in $\widehat{\boldsymbol{y}}_j x_i$ and $\widehat{\boldsymbol{y}}_i x_j$ can be linearized exactly using standard McCormick relaxations [40].

It is then straightforward to generalize piecewise McCormick relaxations to multilinear terms, and we use the following notation to denote these relaxations

$$\left\langle \prod_{k=1}^{|K_t|} x_k \right\rangle^{MC(\mathcal{I})} = \left\langle \left\langle \langle x_1, x_2 \rangle^{MC(\mathcal{I})}, \ldots, x_{|K_t|-1} \right\rangle^{MC(\mathcal{I})}, x_{|K_t|} \right\rangle^{MC(\mathcal{I})}.$$

We also note that the McCormick relaxation is a special case of the piecewise McCormick relaxation when $\mathcal{I}_i = \{\langle x_i^L, x_i^U \rangle\}$.

These relaxations can also be encoded using $\log(|\mathcal{I}_i|)$ binary variables [15,27,52] or variations of special order sets (SOS1, SOS2). For an ease of exposition, we do not present the details of the log-based formulation in this paper. However, later in the results section, we do compare the effectiveness of SOS1 formulations with respect to the linear representation of binary variables.

Piecewise relaxation of a quadratic term Without loss of generality,[1] assume a univariate monomial takes the form x_i^2. Though, we restrict our discussion to a univariate monomial, similar extensions hold true for a multivariate monomial by applying a sequence of relaxations

---

[1] In the case of a higher order univariate monomial, i.e., x_i^5, apply a reduction of the form $x_i^2 x_i^2 x_i \Rightarrow \tilde{x}_i^2 x_i \Rightarrow \tilde{\tilde{x}}_i x_i$.

on the respective univariate monomials. Given partitions in $\mathcal{I}_i$, the piecewise, convex relaxation (see Fig. 1b), denoted by $\widehat{x}_i \in \langle x_i \rangle^{MC_q(\mathcal{I})}$, takes the form:

$$\widehat{x}_i \geqslant x_i^2, \tag{5a}$$

$$\widehat{x}_i \leqslant \left((x_i^l \cdot \widehat{y}_i) + (x_i^u \cdot \widehat{y}_i) \right) x_i - (x_i^l \cdot \widehat{y}_i)(x_i^u \cdot \widehat{y}_i) \tag{5b}$$

$$\widehat{y}_i \cdot \mathbf{1} = 1 \tag{5c}$$

$$\widehat{y}_i \in \{0, 1\}^{|\mathcal{I}_i|} \tag{5d}$$

Once again, $(x_i^l \cdot \widehat{y}_i)(x_i^u \cdot \widehat{y}_i)$ is rewritten as $x_i^l(\widehat{y}_i \widehat{y}_i^T)x_i^u$, where $\widehat{y}_i \widehat{y}_i^T$ is a symmetric matrix with binary product entries (squared binaries on diagonal). Hence, it is sufficient to linearize the entries of the upper triangular matrix with exact representations. We also note again that the unpartitioned relaxation is a special case where $\mathcal{I}_i = \{\langle x_i^L, x_i^U \rangle\}$.

Lemma 1 $\langle x_i \rangle^{MC_q(\mathcal{I})} \subset \langle x_i, x_i \rangle^{MC(\mathcal{I})}$.

Proof Given $\mathcal{I}_i$ for variable x_i, $\langle x_i, x_i \rangle^{MC(\mathcal{I})}$ is given by the following constraints:

$$\widehat{x}_i \geqslant 2(x_i^l \cdot \widehat{y}_i)x_i - (x_i^l \cdot \widehat{y}_i)^2 \tag{6a}$$

$$\widehat{x}_i \geqslant 2(x_i^u \cdot \widehat{y}_i)x_i - (x_i^u \cdot \widehat{y}_i)^2 \tag{6b}$$

$$\widehat{x}_i \leqslant \left((x_i^l \cdot \widehat{y}_i) + (x_i^u \cdot \widehat{y}_i) \right) x_i - (x_i^l \cdot \widehat{y}_i)(x_i^u \cdot \widehat{y}_i) \tag{6c}$$

$$\widehat{y}_i \cdot \mathbf{1} = 1, \ \widehat{y}_i \in \{0, 1\}^{|\mathcal{I}_i|} \tag{6d}$$

First, we claim that any point in $\langle x_i \rangle^{MC_q(\mathcal{I})}$ also lies in $\langle x_i, x_i \rangle^{MC(\mathcal{I})}$. This is trivial to observe since Eqs. (6a) and (6b) are outer approximations of Eq. (5a) at the partition points. To prove $\langle x_i \rangle^{MC_q(\mathcal{I})}$ is a strict subset of $\langle x_i, x_i \rangle^{MC(\mathcal{I})}$, we need to produce a point in $\langle x_i, x_i \rangle^{MC(\mathcal{I})}$ that is not satisfied by $\langle x_i \rangle^{MC_q(\mathcal{I})}$.

Consider the family of points

$$x_i = \frac{1}{2} \left(x_i^l \cdot e_j + x_i^u \cdot e_j \right), \ \widehat{x}_i = \left(x_i^l \cdot e_j \right) \left(x_i^u \cdot e_j \right) \quad \forall j \in 1, \dots, |\mathcal{I}|$$

where, e_j is a unit vector whose jth component takes a value 1. This family of points is satisfied by $\langle x_i, x_i \rangle^{MC(\mathcal{I})}$ and are not contained in $\langle x_i \rangle^{MC_q(\mathcal{I})}$, completing the proof. □

Given these definitions, we use $\mathcal{P}^{\mathcal{I}}$ to denote the piecewise relaxation of $\mathcal{P}$ for a given $\mathcal{I}$, where all the nonlinear monomial terms are replaced with their respective piecewise convex relaxations. More formally,

$$\mathcal{P}^{\mathcal{I}} : \ \underset{x, y}{\text{minimize}} \quad f^{\mathcal{I}}(x, y)$$
$$\text{subject to} \quad g^{\mathcal{I}}(x, y) \leqslant 0,$$
$$h^{\mathcal{I}}(x, y) \leqslant 0, \tag{7}$$
$$x_i^l \cdot \widehat{y}_i \leqslant x_i \leqslant x_i^u \cdot \widehat{y}_i, \quad \forall \, i = 1 \dots n$$
$$y, \widehat{y} \in \{0, 1\}$$

where, $f^{\mathcal{I}}$, $g^{\mathcal{I}}$ and $h^{\mathcal{I}}$ inherit the above defined piecewise relaxations should the functions be nonlinear. Also, we let $f^{\mathcal{I}}(\sigma)$ denote the objective value of a feasible solution, σ, to $\mathcal{P}^{\mathcal{I}}$.

3 Adaptive multivariate partitioning algorithm

This section details the Adaptive Multivariate Partitioning (AMP) algorithm to compute global[2] optimal solutions to MINLPs.

The effectiveness of AMP stems from the observation that the local optimal solutions found by the local solvers are often global optimum or are very close to the global optimum solution on standard benchmark instances. This observation was also made in the literature for the optimal power flow problem in power grids [20,25,29]. AMP exploits this structure and adds sparse, spatial partitions to the variable domains around the local optimal solution. It is important to note that though the partitions are dynamically added around the local optimal point (in the initial iterations), the AMP algorithm *does not* discount the fact that the global optimal solution can potentially lie in sparser regions and will eventually partition the domains which contain the global optimum.

A flow-chart informally describing the steps of AMP and a formal pseudo-code for AMP are given in Fig. 2 and Algorithm 1, respectively. AMP consists of two main components. The first component is a presolve (see lines $1 - 4$). The presolve component of the algorithm is sub-divided into four parts: (i) computing an initial feasible solution, $\overline{\sigma}$, (line 1), (ii) creating an initial set of partitions, $\mathcal{I}$, (iii) sequential OBBT (line 3), and (iv) computing an initial lower bound, $\underline{\sigma}$, using the relaxations detailed in Sect. 2 (line 4).

The second component of AMP is the main loop (lines 5–10) that updates the upper bound, $\overline{\sigma}$, and the lower bound, $\underline{\sigma}$, of $\mathcal{P}$, until either of the following conditions are satisfied: the bounds are within ϵ or the computation time exceeds the limit. At each iteration of the main loop, the partitions are refined and the corresponding piecewise convex relaxation is solved to obtain a lower bound (lines 6 and 7). Similarly the upper bound is obtained using a local solver and updated if it improves the best upper bound computed thus far (lines 8 and 9). In the following sections, we discuss each step of the algorithm in detail.

Algorithm 1 Global optimization using AMP algorithm

Input: $\mathcal{P}$
1: $\overline{\sigma} \leftarrow \text{SOLVE}(\mathcal{P})$ ▷ Compute local optimal solution
2: $\mathcal{I} \leftarrow \text{INITIALIZEPARTITIONS}(\mathcal{P}, \overline{\sigma})$ ▷ Initialize variable partitions
3: $x^l, x^u \leftarrow \text{TIGHTENBOUNDS}(\mathcal{P}^{\mathcal{I}}, \overline{\sigma})$ ▷ Sequential OBBT
4: $\underline{\sigma} \leftarrow \text{SOLVE}(\mathcal{P}^{\mathcal{I}})$ ▷ Initial lower bound computation
5: **while** $\left(\frac{f(\overline{\sigma}) - f^{\mathcal{I}}(\underline{\sigma})}{f^{\mathcal{I}}(\underline{\sigma})} > \epsilon \right)$ and (Time < TimeOut) **do**
6:      $\mathcal{I} \leftarrow \text{REFINEPARTITIONS}(\mathcal{P}^{\mathcal{I}}, \underline{\sigma})$ ▷ Adaptive partition refinement
7:      $\underline{\sigma} \leftarrow \text{SOLVE}(\mathcal{P}^{\mathcal{I}})$ ▷ Compute new lower bound
8:      $\widehat{\sigma} \leftarrow \text{SOLVE}(\mathcal{P}, \underline{\sigma})$ ▷ Compute new local optimum
9:      $\overline{\sigma} \leftarrow \arg\min_{\sigma \in \overline{\sigma} \cup \widehat{\sigma}} f(\sigma)$ ▷ Update upper bound
10: **end while**
Output: $\underline{\sigma}, \overline{\sigma}$

3.1 Presolve

The first step of the presolver of AMP is to compute a local optimal solution, $\overline{\sigma}$, to the MINLP (line 1 of Algorithm 1) . This is done using off-the-shelf, open-source solvers that

---

2 Global optimum is defined numerically by a tolerance, ϵ.
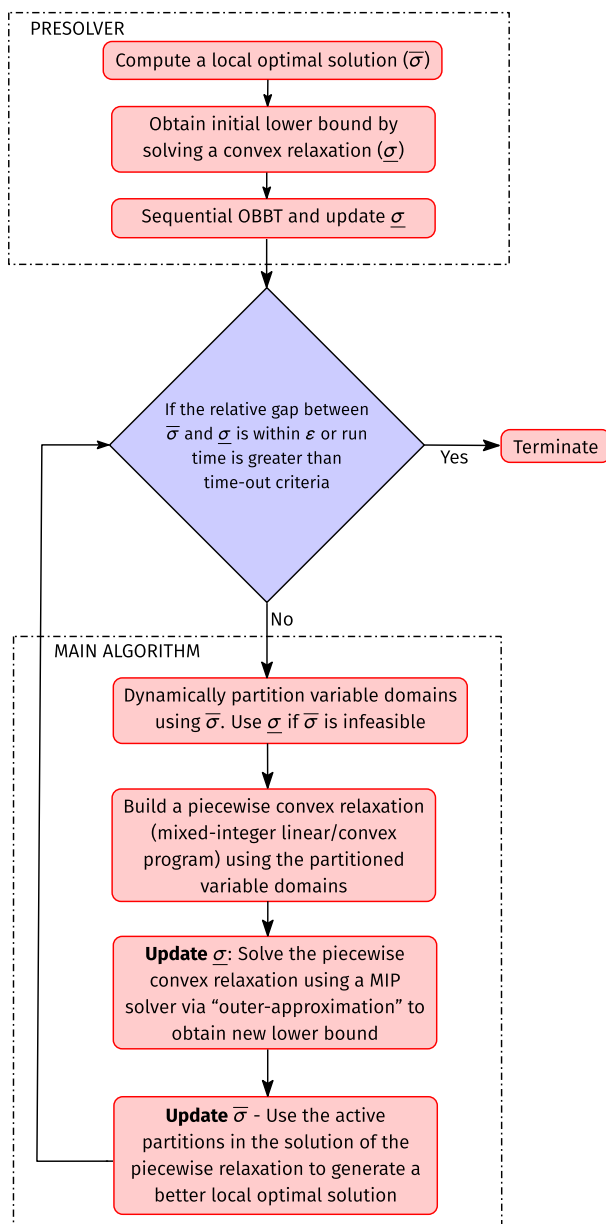
Fig. 2 Flow-chart describing the overall structure of AMP. The flow chart assumes that the MINLP is a feasible minimization problem

use primal-dual interior point methods in conjunction with a branch-and-bound search tree to handle integer variables. This local solution, $\overline{\sigma}$, is further used to initialize the partitions, $\mathcal{I}$ (line 2). When the local solver reports infeasibility, we set the initial value of $f(\overline{\sigma}) = \infty$ (line 5) and use the solution obtained by solving the unpartitioned convex relaxation of $\mathcal{P}$ to

initialize $\mathcal{I}$. In the subsequent sections we detail the partition initialization schemes and the sequential OBBT algorithm in lines 2 and 3 of Algorithm 1.

3.1.1 Partition initialization scheme and sequential OBBT

This section details the algorithm used in lines 2 and 3 of AMP's presolve i.e. INITIALIZEPAR-TITIONS($\mathcal{P}, \overline{\sigma}$) and TIGHTENBOUNDS($\mathcal{P}^{\mathcal{I}}, \overline{\sigma}$), respectively. The sequential OBBT procedure implemented in these functions is one of the key features of AMP. In many engineering applications there is little or no information about the lower and upper bounds ($\boldsymbol{x}^L, \boldsymbol{x}^U$) of the decision variables in the problem. Even when known, the gap between the bounds is often large and weaken relaxations. In practice, replacing the original bounds with tighter bounds can (sometimes) dramatically improve the quality of these relaxations. The basic idea of OBBT is the derivation of (new) valid bounds to improve the relaxations. Though, OBBT is a well-known procedure used in global optimization, the key difference is that we apply OBBT sequentially by using *mixed-integer models to tighten the bounds*.

Algorithm 2 Partition Initialization Scheme

1: **function** INITIALIZEPARTITIONS($\mathcal{P}, \overline{\sigma}$)
2:     **for** $i \in 1 \ldots n$ **do**
3:         $\mathcal{I}_i \leftarrow \{\langle x_i^L, x_i^U \rangle\}$
4:     **end for**
5:     **if** Bound-tightening without partitions **then**
6:         **return** $\mathcal{I}$
7:     **else** $\triangleright$ Bound-tightening with partitions
8:         **return** REFINEPARTITIONS($\mathcal{P}^{\mathcal{I}}, \overline{\sigma}$)
9:     **end if**
10: **end function**

For the sequential OBBT algorithm we present two procedures. First, OBBT without partitions (BT), which is equivalent to partitioning with a single partition. Second, partition-based OBBT (PBT) which uses the standard partitioning approach described in Sect. 3.2.1. For simplicity, we drop the term 'optimization-based (OB)' in the acronym. The BT uses convex optimization problems to tighten the bounds, while the PBT uses convex MIPs to tighten the bounds. The two procedures differ in the initial set of partitions, $\mathcal{I}$, used in the bound-tightening process. Hence, we first present two partition initialization schemes, one for the BT and another for the PBT, respectively. For bound-tightening without partitions, as the name suggests, the partition initialization scheme does not partition the variable domains i.e., $\mathcal{I}_i \leftarrow \{\langle x_i^L, x_i^U \rangle\}$ for every $i = 1, \ldots, n$ (line 6 of Algorithm 2). In the case of PBT, the partition initialization scheme initializes three partitions around the local optimal solution, $\overline{\sigma}$ with a user parameter $\Delta > 1$ (line 8 of Algorithm 2 and Sect. 3.2.1). An illustration of the partitions added to a variable $x \in [x^L, x^U]$, whose value at the local optimal solution is $\overline{\sigma}(x)$, is shown in the Figure 3. If no local solution is obtained in line 1 of Algorithm 1, the solution obtained by solving the unpartitioned convex relaxation of $\mathcal{P}$ is used in place of $\overline{\sigma}(x)$.[3]

Once the initial set of partitions is computed, the sequential OBBT procedure iteratively computes new bounds by solving a modified version of $\mathcal{P}^{\mathcal{I}}$ (Algorithm 3). Each iteration of the sequential OBBT algorithm proceeds as follows: for each continuous variable x_i in

---

[3] To keep the algorithm notation simple, this detail is omitted from Algorithm 2.

Fig. 3 Partition initialization scheme for PBT

$$\overline{\sigma}(x) - \frac{x^U - x^L}{\Delta} \qquad \overline{\sigma}(x) + \frac{x^U - x^L}{\Delta}$$

$$x^L \qquad \overline{\sigma}(x) \qquad x^U$$

$$x^L \leqslant x \leqslant x^L$$
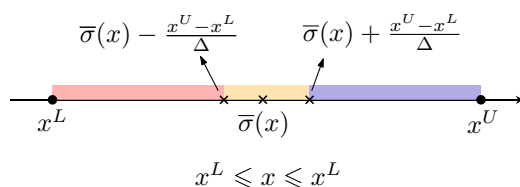
$\mathcal{P}$ appearing in the nonconvex terms, two problems $\mathcal{P}_l^{\mathcal{I}}$ and $\mathcal{P}_u^{\mathcal{I}}$ are solved, where x_i is minimized and maximized, respectively, i.e.,

$$\mathcal{P}_l^{\mathcal{I}}, \mathcal{P}_u^{\mathcal{I}} : \quad \underset{\boldsymbol{x}, \boldsymbol{y}, \widehat{\boldsymbol{y}}}{\text{minimize}} \ \pm x_i \tag{8a}$$

$$\text{subject to} \ f^{\mathcal{I}}(\boldsymbol{x}, \boldsymbol{y}) \leq f(\overline{\sigma}) \tag{8b}$$

$$g^{\mathcal{I}}(\boldsymbol{x}, \boldsymbol{y}) \leq 0, \tag{8c}$$

$$h^{\mathcal{I}}(\boldsymbol{x}, \boldsymbol{y}) = 0, \tag{8d}$$

$$\boldsymbol{x}_i^l \cdot \widehat{\boldsymbol{y}}_i \leq x_i \leq \boldsymbol{x}_i^u \cdot \widehat{\boldsymbol{y}}_i, \quad \forall \, i = 1 \ldots n \tag{8e}$$

$$\boldsymbol{y}, \widehat{\boldsymbol{y}} \in \{0, 1\} \tag{8f}$$

where $\pm x_i$ in formulation (8) denotes two optimization problems, where x_i and $-x_i$ are individually minimized (lines 6–7 of Algorithm 3). In both cases, a constraint that bounds the original objective function of $\mathcal{P}^{\mathcal{I}}$ with a best known feasible solution $\overline{\sigma}$ [Eq. (8b)] is added when an initial feasible solution is available. Inclusion of this constraint on the objective is referred to as optimality/optimization-based bound-tightening in the literature. The OBBT algorithm stops when cumulative bound improvement between successive iterations, measured in terms of the ℓ^∞ norm, falls below specified tolerance values (line 3 of Algorithm 3). We note that the sequential OBBT algorithm is naturally parallelizable as all the optimization problems are independently solvable.

Algorithm 3 Sequential bound-tightening of $\boldsymbol{x}$

1: **function** TIGHTENBOUNDS($\mathcal{P}^{\mathcal{I}}, \overline{\sigma}$)
2: $\quad \widehat{\boldsymbol{x}}^l = \widehat{\boldsymbol{x}}^u \leftarrow \boldsymbol{0}$
3: $\quad$ **while** $||\boldsymbol{x}^l - \widehat{\boldsymbol{x}}^l||_\infty > \epsilon^l$ and $||\boldsymbol{x}^u - \widehat{\boldsymbol{x}}^u||_\infty > \epsilon^u$ **do**
4: $\quad\quad \widehat{\boldsymbol{x}}^l \leftarrow \boldsymbol{x}^l, \ \widehat{\boldsymbol{x}}^u \leftarrow \boldsymbol{x}^u$
5: $\quad\quad$ **for** $i = 1, \ldots, n$ **do**
6: $\quad\quad\quad \sigma_i^l \leftarrow \text{SOLVE}(\mathcal{P}_l^{\mathcal{I}})$
7: $\quad\quad\quad \sigma_i^u \leftarrow \text{SOLVE}(\mathcal{P}_u^{\mathcal{I}})$
8: $\quad\quad\quad \boldsymbol{x}_i^l \leftarrow \max(\sigma_i^l(x_i), \boldsymbol{x}_i^l), \ \boldsymbol{x}_i^u \leftarrow \min(\sigma_i^u(x_i), \boldsymbol{x}_i^u)$
9: $\quad\quad$ **end for**
10: $\quad$ **end while**
11: $\quad$ **return** $\boldsymbol{x}^l, \boldsymbol{x}^u$.
12: **end function**

Remark 1 OBBT procedures, usually referred to as domain reduction techniques, are well known in the global optimization literature [22,42]. Our algorithm generalizes these techniques by performing optimality-based bound-tightening iteratively based on MIP-formulations (with piecewise convex relaxations) until convergence to a fixed point is achieved.

3.2 Main algorithm

The main loop of AMP, shown in lines 5–10 in Algorithm 1 and main algorithm block of the flow chart in Fig. 2, performs three main operations:

1. First, the variable domains are refined adaptively, in a non-uniform manner (line 6 of Algorithm 1).
2. A piecewise convex relaxation is constructed using the partitioned domain, which is solved via outer-approximation to obtain an updated lower bound (line 7 of Algorithm 1)
3. A local solve of the original, MINLP $\mathcal{P}$, with the variables bounds restricted to the partitions obtained by the lower bounding solution, is performed to obtain a new local optimal solution. If this solution is better than the current incumbent, then the incumbent is updated (lines 8 and 9 of Algorithm 1).

In the following subsections, we detail each of the three operations involved in the main loop of AMP.

3.2.1 Variable domain partitioning

One of the core contributions of the AMP algorithm is the adaptive and non-uniform variable partitioning scheme. This part of the algorithm determines how variables in nonconvex terms of the original MINLP are partitioned. Existing approaches partition each variable domain uniformly into a finite number of partitions and the number of partitions increases with the number of iterations [7,12,18,19]. While this is a straight-forward approach for partitioning the variable domains, it potentially creates huge number of partitions far away from the global optimal value of each variable, i.e., many of the partitions are not useful. Though there have been methods, including sophisticated bound propagation techniques and logarithmic encoding to alleviate this issue, they do not scale to large-scale MINLPs. Instead, our approach successively tightens the relaxations with sparse, non-uniform partitions. This approach focuses partitioning on regions of the variable domain that appear to influence optimality the most. These regions are defined by a solution, σ, that is typically the lower bound solution at the current iteration.

The adaptive domain partitioning algorithm of AMP refines the variable partitions at a given iteration with a user parameter $\Delta > 1$. Δ is used to control the size and number of the partitions and influences the rate of convergence of the overall algorithm. The algorithm is similar to the algorithm used for initializing the partitions in Sect. 3.1.1 (see Fig. 3). It differs by using the lower bound solution obtained by solving the piecewise relaxations to refine the partition. Again, for the sake of clarity, Fig. 4 illustrates the bivariate partition refinement for a bilinear term produced by the adaptive domain partitioning algorithm at successive iterations of the main algorithm. Figure 1a geometrically illustrates the tightening of piecewise convex envelopes induced by the adaptive partitioning scheme. The pseudo-code of the domain partitioning algorithm is given in Algorithm 4. The algorithm takes the current variable partitions and a lower bound solution as input and outputs a refined set of partitions for each variable. It first identifies, in line 3 of Algorithm 4, the partition where the lower bound solution is located (active partition) and splits that partition into three new partitions, whose sizes are defined by Δ and the size of the active partition (lines 7–11 in Algorithm 4).

Exhaustiveness of variable domain partitioning scheme Exhaustiveness of a partitioning scheme is one of the important requirements for any global optimization algorithm [23]. The

(a) Iteration k

(b) Iteration $k+1$
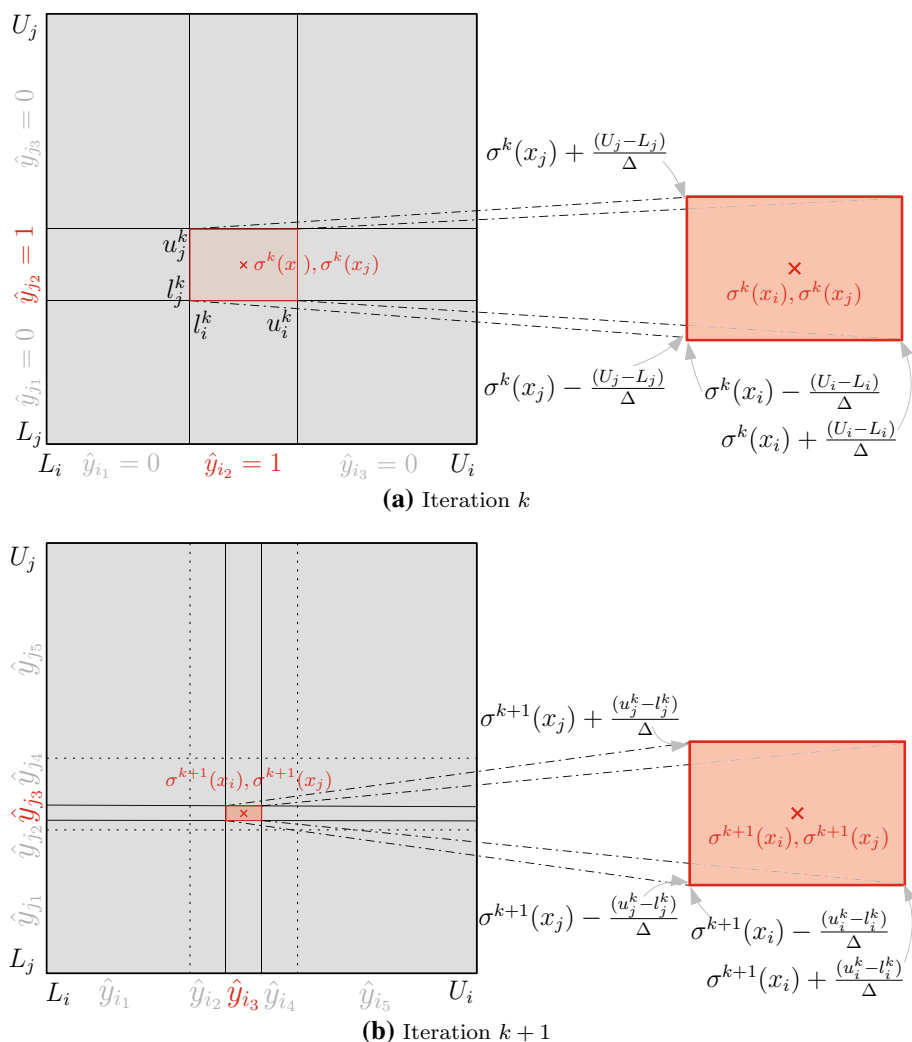
Fig. 4 Adaptive partitioning strategy for a bilinear function $x_i x_j$ as described in Algorithm 4. Red and gray colored boxes represent active and inactive partitions, respectively. k and $k+1$ refer to successive solutions used to partition the active partition. (Color figure online)

exhaustiveness of the adaptive domain partitioning scheme is built into AMP through the lines 13–14 in Algorithm 4. These conditions ensure that the AMP algorithm does not get stuck at a particular region of the variable. Instead, it guarantees that the largest partition outside the active partition is further refined when the active partition's width is less than a partition-width tolerance value, ϵ^p. Here, the largest inactive partition of a given variable is analogous to the largest unexplored domain for that variable. Thus, the partitioning scheme in the AMP algorithm satisfies the desirable exhaustiveness property.

Algorithm 4 Variable Domain Partitioning

1: **function** REFINEPARTITIONS($\mathcal{P}^{\mathcal{I}}, \sigma$)
2:     **for** $i \in 1 \ldots n$ **do**
3:         $k \leftarrow \arg \max \sigma(\hat{y}_i^k)$ ▷ Identifying the active partition for variable i
4:         $\langle l_i, u_i \rangle \leftarrow \mathcal{I}_i^k$
5:         $\xi_i \leftarrow \frac{u_i - l_i}{\Delta}$
6:         **if** $\xi_i > \epsilon^p$ **then**   ▷ Active partition is split into three partitions
7:             $\gamma_1 \leftarrow \min(l_i, \max(\sigma(x_i) - \xi_i, x_i^L))$
8:             $\gamma_2 \leftarrow \max(l_i, \sigma(x_i) - \xi_i)$
9:             $\gamma_3 \leftarrow \min(u_i, \sigma(x_i) + \xi_i)$
10:            $\gamma_4 \leftarrow \max(u_i, \min(\sigma(x_i) + \xi_i, x_i^U))$
11:            $\mathcal{I}_i \leftarrow (\mathcal{I}_i \setminus \langle l_i, u_i \rangle) \cup \langle \gamma_1, \gamma_2 \rangle \cup \langle \gamma_2, \gamma_3 \rangle \cup \langle \gamma_3, \gamma_4 \rangle$
12:         **else**   ▷ Partitioning of largest inactive partition
13:            $\langle l_i, u_i \rangle \leftarrow \arg \max_{\mathcal{I}_i} u_i - l_i$
14:            $\mathcal{I}_i \leftarrow (\mathcal{I}_i \setminus \langle l_i, u_i \rangle) \cup \langle l_i, l_i + \frac{u_i - l_i}{2} \rangle \cup \langle l_i + \frac{u_i - l_i}{2}, u_i \rangle$
15:         **end if**
16:     **end for**
17:     **return** $\mathcal{I}$
18: **end function**

3.2.2 Computing lower and upper bounds

Once the variable partitions are refined, the main loop of the AMP algorithm constructs and solves a piecewise convex relaxation, $\mathcal{P}^{\mathcal{I}}$ (line 7 of Algorithm 1). Each $\mathcal{P}^{\mathcal{I}}$ is a convex MIP where all the constraints are either linear or second-order cones (SOCs). Theoretically, $\mathcal{P}^{\mathcal{I}}$ can be solved by off-the-shelf mixed-integer, conic solvers. However, initial computational experiments suggested that several moderately sized problems with SOC constraints were difficult to solve, even with state-of-the-art solvers. It was the case that either the solver convergence was very slow or the solve terminated with a numerical error. To circumvent this issue and solve these convex MIPs in a computationally efficient manner, the SOC constraints in the convex MIP were outer-approximated via first-order approximations, using the lazy-callback feature in the MIP solvers.

To obtain a new local optimal solution in the main loop of AMP (line 8 of Algorithm 1), the feasible solution is obtained by solving the NLP, $\mathcal{P}^u$, shown in Eq. (9). $\mathcal{P}^u$ is constructed at each iteration of the main loop using the original MINLP, $\mathcal{P}$, and the lower bound solution computed at that iteration, $\underline{\sigma}$.

$$\mathcal{P}^u : \quad \underset{x, y}{\text{minimize}} \quad f(x, y) \tag{9a}$$

$$\text{subject to} \quad g(x, y) \leq 0, \tag{9b}$$

$$h(x, y) = 0, \tag{9c}$$

$$x_i^l \cdot \underline{\sigma}(\hat{y}_i) \leq x_i \leq x_i^u \cdot \underline{\sigma}(\hat{y}_i), \quad \forall \, i = 1 \ldots n \tag{9d}$$

$$y = \underline{\sigma}(y) \tag{9e}$$

where constraint (9d) forces the variable assignments into the partition defined by the current lower bound. Constraint (9e) fixes all the original binary variables to the lower bound solution. $\mathcal{P}^u$ is then solved to local optimality using a local solver. The motivation for this approach is based on empirical observations that the relaxed solution is often very near to the global optimum solution and that the NLP is solved fast once all binary variables are fixed to constant

values. $\mathcal{P}^u$ is essentially a projection of the relaxed solution (lower bound solution, $\underline{\sigma}$) back onto a near point in the feasible region of $\mathcal{P}$; this approach is often used for recovering feasible solutions [7]. In the forthcoming Lemma 2, we claim that the value of the objective function of the piecewise convex relaxation at each iteration monotonically increases to the global optimum solution with successive partition refinements.

Lemma 2 *Let $\underline{\sigma}^{\mathcal{I}}$ denote the optimal solution to the formulation $\mathcal{P}^{\mathcal{I}}$ and let σ^* denote the global optimal solution to $\mathcal{P}$. Then, $f^{\mathcal{I}}(\underline{\sigma}^{\mathcal{I}})$ monotonically increases to $f(\sigma^*)$ as $|\mathcal{I}_i| \to \infty$ for every $i = 1, \ldots, n$.*

Proof Without loss of generality, we assume $\mathcal{P}$ is feasible and restrict our discussion to bilinear terms. Let $x_i x_j$ be a bilinear term. Given a finite set of partitions, that is, $1 \leqslant |\mathcal{I}_i|, |\mathcal{I}_j| < \infty$, there always exists a partition in $\mathcal{I}_i$ and $\mathcal{I}_j$ that is active in the solution to $\mathcal{P}^{\mathcal{I}}$. Let the active partitions have lengths $\epsilon_i^l + \epsilon_i^u$ and $\epsilon_j^l + \epsilon_j^u$, respectively. Given the exhaustiveness property of the adaptive partitioning scheme discussed in Sect. 3.2.1, assume the active partition contains the global optimum solution $\sigma^*(\mathbf{x}^*, \mathbf{y}^*)$.[4] Then, we have

$$x_i^* - \epsilon_i^l \leqslant x_i \leqslant x_i^* + \epsilon_i^u, \quad x_j^* - \epsilon_j^l \leqslant x_j \leqslant x_j^* + \epsilon_j^u.$$

For these active partitions, the McCormick constraints in (4a) and (4c) linearize $x_i x_j$ as follows:

$$
\begin{aligned}
\widehat{x_{ij}} &\geqslant (x_i^* - \epsilon_i^l)x_j + (x_j^* - \epsilon_j^l)x_i - (x_i^* - \epsilon_i^l)(x_j^* - \epsilon_j^l) \\
&= (x_i^* x_j + x_j^* x_i - x_i^* x_j^*) + \underbrace{\epsilon_i^l(x_j^* - x_j) + \epsilon_j^l(x_i^* - x_i) - \epsilon_i^l \epsilon_j^l}_{\mathcal{E}(\epsilon_i^l, \epsilon_j^l)}
\end{aligned}
\tag{10}
$$

and

$$
\begin{aligned}
\widehat{x_{ij}} &\leqslant (x_i^* - \epsilon_i^l)x_j + (x_j^* + \epsilon_j^u)x_i - (x_i^* - \epsilon_i^l)(x_j^* + \epsilon_j^u) \\
&= (x_i^* x_j + x_j^* x_i - x_i^* x_j^*) + \underbrace{\epsilon_i^l(x_j^* - x_j) + \epsilon_j^u(x_i - x_i^*) + \epsilon_i^l \epsilon_j^u}_{\mathcal{E}(\epsilon_i^l, \epsilon_j^u)}
\end{aligned}
\tag{11}
$$

where, $\mathcal{E}(\epsilon_i^l, \epsilon_j^l) < 0$ and $\mathcal{E}(\epsilon_i^l, \epsilon_j^u) > 0$ are the error terms of the under- and over-estimator, respectively. It is trivial to observe that the error terms themselves constitute the McCormick envelopes of $(x_i^* - x_i)(x_j^* - x_j)$ if one were to linearize this product. Further, observing that the error terms, as described above, are parametrized by the size of the active partition containing the global optimum point, given by $\epsilon_i^l + \epsilon_i^u$ and $\epsilon_j^l + \epsilon_j^u$, for variables x_i and x_j, respectively, we now derive the analytic forms of these terms as a function of the total number of partitions for the adaptive partitioning case.

Line 6 in Algorithm 1 ensures that the partitions created during the $(k+1)^{th}$ iteration of the main loop for either of the variables, x_i or x_j, is a subset of the partitions created for the corresponding variables in the kth iteration. Also, for a kth iteration of an adaptive refinement step for variables x_i and x_j, we assume that at most $3 + 2(k-1)$ partitions exist within the given variable bounds (L_i, U_i) and (L_j, U_j), respectively. Thus, the length of the above mentioned active partition which contains the global solution is given by

$$\epsilon_i^l + \epsilon_i^u = \frac{U_i - L_i}{\Delta^{\frac{|\mathcal{I}_i|-1}{2}}}, \quad \epsilon_j^l + \epsilon_j^u = \frac{U_j - L_j}{\Delta^{\frac{|\mathcal{I}_j|-1}{2}}}.$$

---

[4] Exhaustiveness of the partitioning scheme implies AMP will eventually partition all other domains small enough such that AMP will pick an active partition with the global optimal whose length is $\leq \epsilon_i^l + \epsilon_i^u$.

Clearly as $|\mathcal{I}_i|$ and $|\mathcal{I}_j|$ approach ∞, the error terms of the McCormick envelopes, $\mathcal{E}(\epsilon_i^l, \epsilon_j^l)$ and $\mathcal{E}(\epsilon_i^l, \epsilon_j^u)$, approach zero, and thus enforcing $x_i = x_i^*$, $x_j = x_j^*$ and $\widehat{x_{ij}} = x_i^* x_j^*$. Therefore, as $|\mathcal{I}_i|$ approaches ∞ for every variable i that is partitioned, $f^{\mathcal{I}}(\underline{\sigma}^{\mathcal{I}})$ approaches the global optimal solution $f(\sigma^*)$.

Also observe that since $\mathfrak{F}(\mathcal{P}^{\mathcal{I}}) \subset \mathfrak{F}(\mathcal{P})$ for any finite $\mathcal{I}$, $f^{\mathcal{I}}(\underline{\sigma}^{\mathcal{I}}) \leqslant f(\sigma^*)$. Here, $\mathfrak{F}(\cdot)$ denotes the feasible space of the formulation of "$\cdot$". Furthermore, the partition set in iteration k is a proper subset of the previous iteration's partitions. This proves the monotonicity of the sequence of values $f^{\mathcal{I}}(\underline{\sigma}^{\mathcal{I}})$ with increasing iterations of the main loop of AMP. $\qquad \square$

4 Computational results

In the remainder of this paper, we refer to AMP as Algorithm 1 without the implementation of line 3, i.e., without any form of sequential OBBT. BT-AMP and PBT-AMP refer to Algorithm 1 implemented with bound-tightening without and with partitions added, respectively. The performance of these algorithms is evaluated on a set of standard benchmarks from the literature with mutlilinear terms. These problems include a small NLP that is used to highlight the differences between the sparse, adaptive approaches and uniform partitioning approaches. The details and sources for each problem instance are shown later in this section in Table 7. In this table, we also mention the continuous variables in mutlilinear terms chosen for partitioning.[5] Ipopt 3.12.8 and Bonmin 1.8.2 are used as local NLP and MINLP solvers for the feasible solution computation in AMP, respectively. MILPs and MIQCQCPs are solved using CPLEX 12.7 (cpx) and/or Gurobi 7.0.2 (grb) with default options and presolver switched on. The outer-approximation algorithm was implemented using the lazy callback feature of CPLEX and Gurobi. Given that the bound-tightening procedure consists of independently solvable problems, 10 parallel threads were used during bound-tightening. In the Appendix we provide a detailed sensitivity analysis of the parameters of AMP.

Every bound-tightening (BT and PBT) problem was solved to optimality (except *meyer15*). For *meyer15*, 0.1% optimality gap was used as a termination criteria because it is a large-scale MINLP. The value of ϵ and the "TimeOut" parameter in Algorithm 1 were set to 0.0001 and 3600 s, respectively. However, for BT and PBT, we did not impose any time limit. Thus, for a fair comparison, we set the time limit for the global solver to be the sum of bound tightening time and 3600 s (denoted by T^+ in the tables). In the results, "TO" indicates that the AMP solve timed-out. All results are benchmarked with BARON 17.1, a state-of-the-art global optimization solver [47,49]. CPLEX 12.7 and Ipopt 3.12.8 are used as the underlying MILP and non-convex local solvers for BARON. JuMP, an algebraic modeling language in Julia [14], was used for implementing all the algorithms and invoking the optimization solvers. All the computational experiments were performed using the high-performance computing resources at the Los Alamos National Laboratory with Intel CPU E5-2660-v3, Haswell micro-architecture, 20 cores (2 threads per core) and 125GB of memory.

---

[5] See [8] for more details on strategies for choosing the variables for partitioning.

4.1 Performance on a small-scale NLP

$$
\begin{aligned}
\textit{NLP1:} \quad &\underset{x_1,\ldots,x_8}{\text{minimize}} && x_1 + x_2 + x_3 \\
&\text{subject to} && 0.0025(x_4 + x_6) - 1 \le 0, \\
&&& 0.0025(-x_4 + x_5 + x_7) - 1 \le 0, \\
&&& 0.01(-x_5 + x_8) - 1 \le 0, \\
&&& 100x_1 - x_1x_6 + 833.33252x_4 - 83333.333 \le 0, \\
&&& x_2x_4 - x_2x_7 - 1250x_4 + 1250x_5 \le 0, \\
&&& x_3x_5 - x_3x_8 - 2500x_5 + 1250000 \le 0, \\
&&& 100 \le x_1 \le 10000, \\
&&& 1000 \le x_2, x_3 \le 10000, \\
&&& 10 \le x_4, x_5, x_6, x_7, x_8 \le 1000
\end{aligned}
$$

In this section, we perform a detailed study of AMP with and without OBBT on *NLP1*, a small-scale, continuous nonlinear program adapted from Problem 106 in [21]. This small problem helps illustrate many of the salient features of AMP. *NLP1* has gained considerable interest from the global optimization literature due its large variable bounds and weak McCormick relaxations. Since this is a challenging problem for uniform, piecewise McCormick relaxations, this problem has been studied in detail in [11,12,50]. The value of the global optimum for *NLP1* is 7049.2479 and the solution is x_i^*, $i = 1, \ldots, 8 = $ [579.307, 1359.97, 5109.97, 182.018, 295.601, 217.982, 286.417, 395.601].

4.1.1 AMP versus uniform partitioning on *NLP1*

Figure 5 compares AMP (without BT/PBT) with a uniform partitioning strategy that is often used in state-of-the-art solvers to obtain global solutions. For a fair comparison, we used the same number of partitions for both methods at every iteration. From Fig. 5a, it is evident that AMP exhibits larger optimality gaps in the first few iterations (134%, 44%, 20%, vs. 97%, 44%, 16%, etc.). However, the convergence rate to global optimum is much faster with AMP (within 190.9 s). This behaviour is primarily attributed to the adaptive addition of partitions around the best-known local solution (also global in this case) instead of spreading them uniformly.

4.1.2 Performance of AMP on *NLP1*

Figure 6 shows the active partitions chosen by every iteration of AMP for $\Delta = 4$. This figure illustrates the active partitions at each iteration of the main loop of AMP and the convergence of each variable partition to its corresponding global optimal value. One of the primary motivations of the adaptive partitioning strategy comes from the observation that every new partition added adaptively refines the regions (hopefully) closer to the global optimum values. In Fig. 6a, this behaviour is clearly evident on all the variables except x_3. Although the initial active partition on x_3 did not contain the global optimum, AMP converged to the global optimum value. The convergence time of AMP to the global optimum using Gurobi was *190.9 s*. The total number of binary partitioning variables is 152 (19 per continuous variable).

(a) Optimality gap

(b) Cumulative time

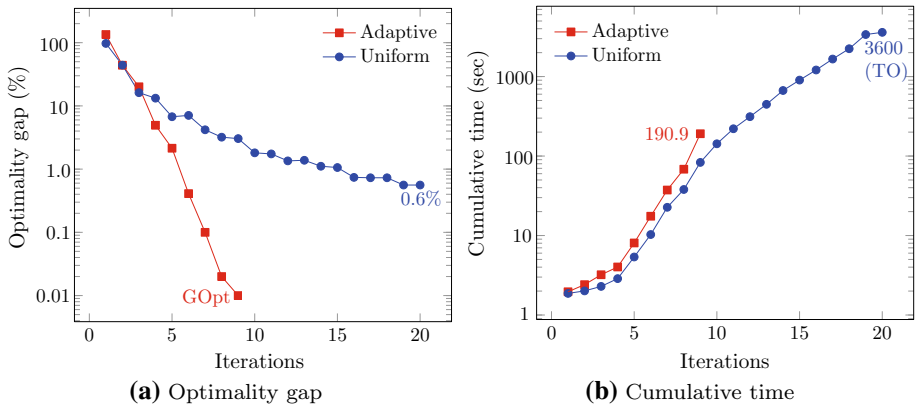**Fig. 5** Performance of AMP ($\Delta = 4$) and uniform partitioning on *NLP1*. Note that the y-axis is on log scale

(a) Variables x_1, x_3

(b) Variables x_2, x_4

(c) Variables x_5, x_6
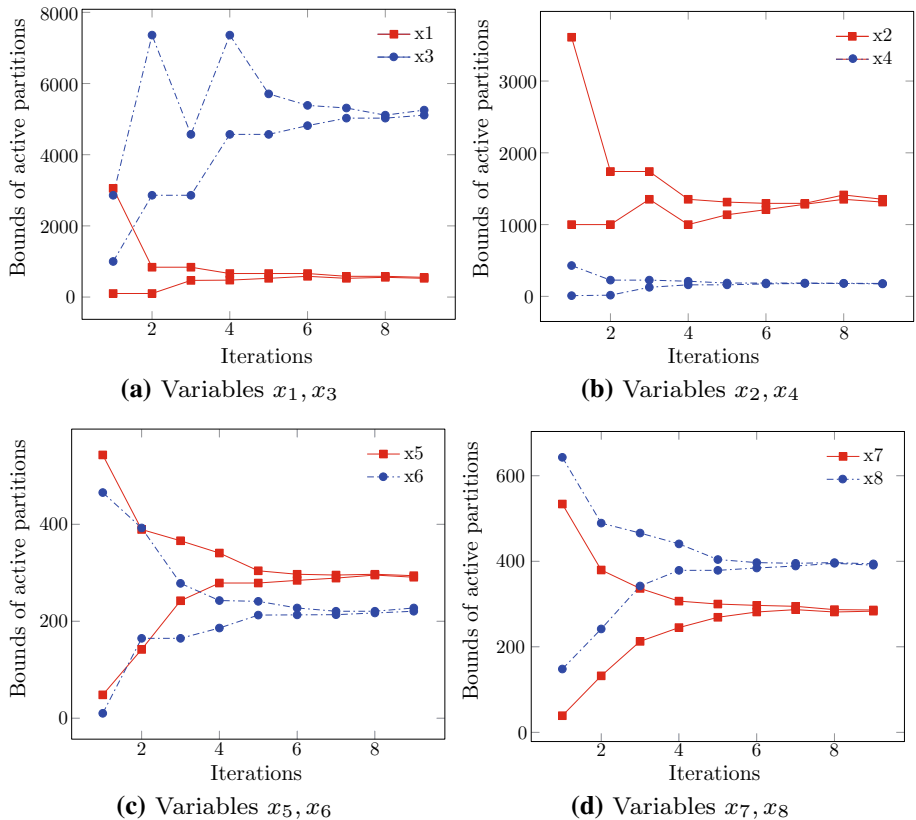
(d) Variables x_7, x_8

Fig. 6 Upper and lower bounds of active partitions chosen by AMP algorithm (without OBBT) for the variables $x_i, i = 1, \ldots, 8$ of *NLP1*

(a) Tightened bounds after each iteration

(b) Time per iteration
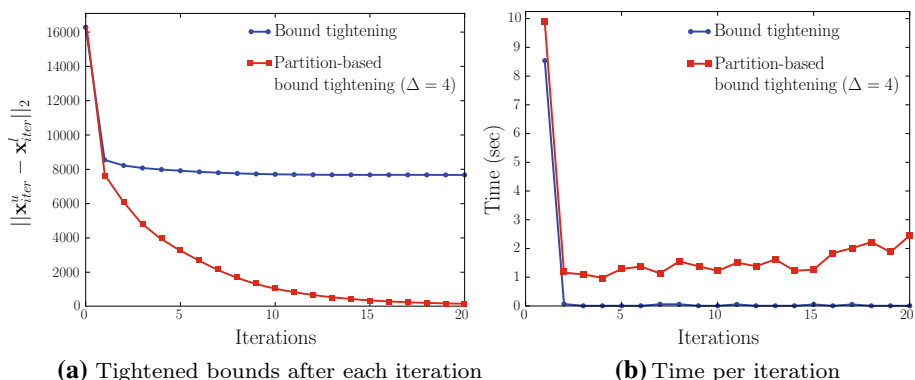
Fig. 7 Performance of sequential BT and sequential PBT techniques on *NLP1*

Table 1 Contracted bounds after applying sequential bound tightening to nlp1

Variable	Original bounds		PBT bounds		#BVars added	
	L	U	L	U	BT, AMP ($\Delta = 4$)	PBT, AMP ($\Delta = 4$)
x_1	100	10, 000	573.1	585.1	0, 14	3, 3
x_2	1000	10, 000	1351.2	1368.5	0, 14	3, 3
x_3	1000	10, 000	5102.1	5117.5	0, 15	3, 3
x_4	10	1000	181.5	182.5	0, 15	3, 3
x_5	10	1000	295.3	296.0	0, 15	3, 3
x_6	10	1000	217.5	218.5	0, 15	3, 3
x_7	10	1000	286.0	286.9	0, 15	3, 3
x_8	10	1000	395.3	396.0	0, 15	3, 3
Total					118	48

4.1.3 Benefits of MIP-based OBBT on *NLP1*

Figure 7 and Table 1 show the effectiveness of sequential BT and sequential PBT techniques on *NLP1*. As expected, in Fig. 7a, the disjunctive polyhedral representation of the relaxed regions in PBT (around the initial local solution) drastically reduce the global bounds on the variables (to almost zero gaps). Figure 7b shows that PBT, even when solving a MILP in every iteration, does not incur too much computational overhead on a small-scale problem like *NLP1*.

A qualitative description of the improved performance is presented in Table 1. The column titled "#BVars added" shows the total number of partitions that were added for each variable for BT-AMP and PBT-AMP. While BT adds no partitions and AMP adds a total of 118 partitions, PBT adds a total of 24 partitions in addition to 24 more partitions during AMP. As is seen in the results in the "PBT bounds" column, the bounds of the variables are tightened to near global optimum values and AMP needs very few additional partitions to prove global optimality within 40 s. In contrast, AMP adds a lot more partitions as the bounds of the variables after BT are not tight enough. Overall, for *NLP1*, it is noteworthy that PBT-AMP outperforms most of the state-of-the-art piecewise relaxation methods developed in the literature.

4.2 Performance of AMP on large-scale MINLPs

In this section we assess the empirical value of adaptive partitioning by presenting results without OBBT. In Table 2, AMP (without OBBT) with CPLEX or Gurobi is compared with BARON. Columns two and three show the run times of Couenne and BARON, respectively, based on the 3600 second time limit. Column four shows the performance of AMP for $\Delta = 8$. Though $\Delta = 8$ is not the ideal setting for every instance, it is analogous to running BARON/Couenne with default parameters. Under the default AMP settings, AMP is faster than BARON and Couenne (with default settings) at finding the best lower bound in 23 out of 34 instances. These results of AMP are the most fair to compare with untuned BARON and Couenne

In column five (tuned Δ and CPLEX), the run times of AMP are much faster than BARON and Couenne in 24 out of 34 instances. Column six (tuned Δ and Gurobi) again indicates that the right choice of Δ speeds up the convergence of AMP drastically. More interestingly, on 21 out of 34 instances, the run times of AMP using Gurobi are substantially better than the run times using CPLEX (column 5).

Table 2 is summarized with a cumulative distribution plot in Fig. 8. Clearly, Figure 8a indicates that AMP is better able to find solutions within a 0.4% optimality gap (even without tuning). Figure 8b provides evidence of the overall strength of AMP. Even when AMP is not the fastest approach, its run times are very similar to BARON. Overall, the performance of AMP is clearly better using Gurobi as the underlying MILP/MIQCQP solver. We did not perform comparative studies of BARON with Gurobi because it cannot currently integrate with Gurobi.

4.3 Performance of AMP with OBBT

We next discuss the performance of AMP when OBBT is added.

4.3.1 Default parameters of Δ

We first consider AMP with OBBT when AMP uses the default parameter of $\Delta = 8$. Table 3, compares AMP with BARON. Column two shows the run times of BARON based on a prescribed time limit. For comparison purposes with AMP, the time limit of BARON is calculated as the sum of 3600 s and the maximum of the run time of BT and PBT. For the purposes of this study, we did not specify a time limit on BT and PBT, though this could be added.

Column three shows the performance of BT-AMP when $\Delta = 8$. While a constant Δ is not the ideal parameter for every instance, BT-AMP with CPLEX is still faster than BARON on 21 out of 32 instances. Similarly, BT-AMP with Gurobi is faster in 25 out of 32 instances. Once again, AMP with Gurobi has significant computational advantages over CPLEX. Instances *blend480*, *blend721*, *blend852*, and *meyer10* demonstrated an order of magnitude improvement. Column four of Table 3 shows the results of PBT-AMP when $\Delta = 10$. Though PBT solves a more complicated, discrete optimization problem at every step of bound-tightening, surprisingly, the total time spent in bound-tightening was typically significantly smaller. This is seen in all instances prefixed with "*ex*", the *util* instance, the *eniplac* instance, the *meanvarx* instance and a few *blend* instances. In general, using partition-based OBBT yields significant improvements in the overall run times and optimality gaps of AMP.

Table 2 Summary of the performance of AMP without OBBT on all instances

Instances	COUENNE		BARON		AMP-cpx Δ = 8		AMP-cpx Δ*			AMP-grb Δ*		
	Gap	T	Gap	T	Gap	T	Δ	Gap	T	Δ	Gap	T
p1	**GOpt**	**0.01**	GOpt	0.02	GOpt	0.26	32	GOpt	0.19	32	GOpt	0.06
p2	**GOpt**	**0.01**	**GOpt**	**0.01**	GOpt	0.05	16	GOpt	0.03	32	GOpt	0.10
fuel	Inf	N/A	**GOpt**	**0.03**	GOpt	0.07	4	**GOpt**	**0.03**	4	GOpt	0.05
ex1223a	**GOpt**	**0.01**	GOpt	0.02	**GOpt**	**0.01**	32	**GOpt**	**0.01**	16	GOpt	0.02
ex1264	GOpt	2.04	GOpt	1.44	GOpt	1.42	16	GOpt	0.90	8	**GOpt**	**0.79**
ex1265	GOpt	5.22	GOpt	13.30	GOpt	0.94	16	**GOpt**	**0.17**	8	GOpt	0.28
ex1266	GOpt	5.37	GOpt	10.81	GOpt	0.27	32	**GOpt**	**0.14**	32	GOpt	0.16
eniplac	GOpt	128.82	GOpt	207.37	GOpt	1.17	32	**GOpt**	**0.68**	32	GOpt	0.75
util	GOpt	14.56	**GOpt**	**0.10**	GOpt	1.21	16	GOpt	0.54	16	GOpt	0.55
meanvarx	GOpt	1.06	**GOpt**	**0.05**	GOpt	290.61	16	GOpt	95.51	16	GOpt	70.09
blend029	GOpt	35.18	GOpt	2.46	GOpt	1.74	32	**GOpt**	**0.74**	32	GOpt	1.00
blend531	3.06	TO	GOpt	111.79	GOpt	185.33	8	GOpt	185.33	16	**GOpt**	**49.76**
blend146	4.19	TO	2.20	TO	2.01	TO	8	2.01	TO	8	**1.60**	**TO**
blend718	156.45	TO	175.10	TO	**GOpt**	**379.58**	8	**GOpt**	**379.58**	8	GOpt	581.68
blend480	102.41	TO	**GOpt**	**326.95**	0.32	TO	32	0.04	TO	16	0.02	TO
blend721	0.60	TO	GOpt	548.90	GOpt	504.74	32	GOpt	256.77	16	**GOpt**	**176.11**
blend852	1.74	TO	0.08	TO	GOpt	750.88	4	**GOpt**	**169.24**	16	GOpt	322.80
wtsM2_05	GOpt	3426.28	**GOpt**	**153.30**	20.08	TO	8	20.08	TO	32	GOpt	386.95
wtsM2_06	31.95	TO	**GOpt**	**228.18**	8.76	TO	4	GOpt	2395.71	32	GOpt	972.20
wtsM2_07	**GOpt**	**68.37**	GOpt	759.96	0.10	TO	8	0.10	TO	16	0.54	TO
wtsM2_08	39.43	TO	388.62	TO	9.82	TO	4	**5.45**	**TO**	4	7.92	TO
wtsM2_09	60.37	TO	Inf	TO	68.58	TO	10	36.47	TO	4	**7.47**	**TO**

Table 2 continued

Instances	COUENNE		BARON		AMP-cpx Δ = 8		AMP-cpx Δ*			AMP-grb Δ*		
	Gap	T	Gap	T	Gap	T	Δ	Gap	T	Δ	Gap	T
wtsM2_10	0.64	TO	76.48	TO	35.88	TO	32	24.95	TO	16	**0.10**	**TO**
wtsM2_11	64.71	TO	107.56	TO	7.88	TO	16	**3.50**	**TO**	4	6.10	TO
wtsM2_12	68.56	TO	85.35	TO	8.07	TO	4	7.46	TO	32	**4.00**	**TO**
wtsM2_13	47.74	TO	54.04	TO	10.06	TO	4	**4.24**	**TO**	8	5.72	TO
wtsM2_14	47.15	TO	46.24	TO	9.02	TO	32	6.34	TO	16	**1.43**	**TO**
wtsM2_15	61.32	TO	Inf	TO	86.64	TO	4	8.81	TO	8	**0.22**	**TO**
wtsM2_16	26.72	TO	47.77	TO	34.46	TO	8	34.46	TO	32	**5.25**	**TO**
lee1	GOpt	46.97	GOpt	145.55	**GOpt**	**13.01**	8	**GOpt**	**13.01**	8	GOpt	13.61
lee2	**GOpt**	**60.43**	GOpt	590.08	0.58	TO	16	0.47	TO	16	0.08	TO
meyer4	Inf	N/A	80.40	TO	GOpt	18.85	4	GOpt	12.50	8	**GOpt**	**5.68**
meyer10	80.88	TO	239.70	TO	GOpt	TO	4	GOpt	452.53	8	**GOpt**	**133.47**
meyer15	Inf	N/A	2850.30	TO	0.59	TO	16	0.31	TO	4	**0.10**	**TO**

Here, we compare the run times of BARON, AMP, AMP with Δ = 8 (cpx), AMP with the best Δ (cpx) and AMP with the best Δ (grb). Values under "Gap" and "T" are in % and seconds, respectively. "Inf" implies that the solver failed to provide a bound within the prescribed time limit. For each instance, the bold face font represents best run time or the best optimality gap (if the solve times out)
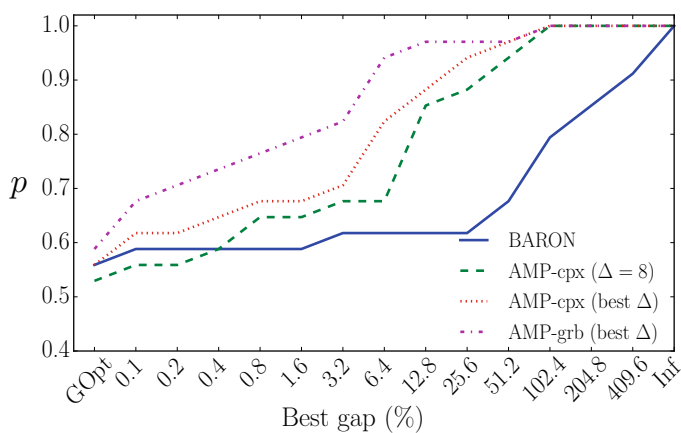
(a) Comparison of best gap
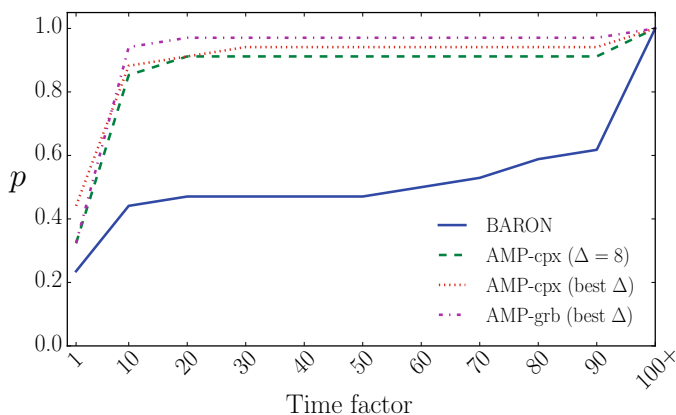
(b) Comparison of best run times

Fig. 8 Performance profiles of AMP (without OBBT) and BARON. In **a** the x axis plots the optimality gap of the algorithms and the y axis plots fraction of instances. Plot **a** tracks the number of instances where an algorithm is able achieve the specified optimality gap. In **b** the x axis denotes the run time ratio of an algorithm with the best run time of any algorithm. The y axis denotes the fraction of instances. Plot **b** tracks the number of times an algorithm's run time is within a specified factor of the best run time of any algorithm. In both figures, higher is better. Overall, AMP performs better than BARON on a p proportion of instances for most gaps and all run times

The bound-tightening procedure also compares favorably with BARON. For example, consider problem *blend852*. Though BARON implements a sophisticated bound-tightening approach that is based on primal and dual formulations, BARON times out with a 0.08% gap. In contrast, BT-AMP with Gurobi converges to the global optimum in 434.7 s and PBT-AMP converges to the global optimum in 78.1 s (an order-of-magnitude improvement). Similar behaviour is observed on the remaining *blend*, *wts* and *meyer* instances. Overall, AMP with Gurobi outperforms BARON on 24 out of 32 instances when a default choice of Δ is used.

Table 3 is summarized with a cumulative distribution plot in Fig. 9. Clearly, Fig. 9a indicates that BT-AMP and PBT-AMP with Gurobi performs better than BARON even without tuning Δ. In Fig. 9a, AMP has a better profile when the optimality gap is $> 0.4\%$. In Fig. 9a,

Table 3 Performance summary of AMP with OBBT on all instances

Instances	BARON		BT	AMP-cpx		AMP-grb		PBT	AMP-cpx		AMP-grb	
	Gap	T	T^+	Gap	T	Gap	T	T^+	Gap	T	Gap	T
fuel	GOpt	0.03	0.01	GOpt	0.03	GOpt	0.03	0.01	GOpt	0.04	GOpt	0.03
ex1223a	GOpt	0.02	19.71	GOpt	0.01	GOpt	0.01	0.30	GOpt	0.01	GOpt	0.02
ex1264	GOpt	1.44	12.47	GOpt	1.77	GOpt	1.32	0.72	GOpt	1.48	GOpt	1.24
ex1265	GOpt	13.3	6.02	GOpt	0.25	GOpt	0.88	1.02	GOpt	0.26	GOpt	0.74
ex1266	GOpt	10.81	13.75	GOpt	0.12	GOpt	0.06	1.30	GOpt	0.04	GOpt	0.06
eniplac	GOpt	207.37	16.04	GOpt	1.13	GOpt	1.36	3.34	GOpt	1.17	GOpt	1.59
util	GOpt	0.10	9.92	GOpt	0.17	GOpt	0.19	0.61	GOpt	0.14	GOpt	0.16
meanvarx	GOpt	0.05	20.69	GOpt	95.23	GOpt	59.33	3.53	GOpt	13.62	GOpt	13.31
blend029	GOpt	2.46	15.05	GOpt	1.04	GOpt	1.56	0.80	GOpt	0.88	GOpt	0.95
blend531	GOpt	111.79	44.67	GOpt	38.60	GOpt	22.56	477.94	GOpt	39.89	GOpt	20.12
blend146	2.20	TO	30.95	24.92	TO	0.10	TO	26.66	24.98	TO	23.69	TO
blend718	175.10	TO	28.76	GOpt	1332.66	GOpt	1335.93	20.8	14.65	TO	GOpt	868.14
blend480	GOpt	326.95	137.62	0.21	TO	GOpt	108.93	1699.18	8.78	TO	GOpt	2466.17
blend721	GOpt	548.9	29.44	GOpt	646.92	GOpt	181.88	23.92	GOpt	93.28	GOpt	112.91
blend852	0.08	TO	41.73	GOpt	749.03	GOpt	392.99	29.40	GOpt	217.62	GOpt	48.79
wtsM2_05	GOpt	153.30	14.82	0.24	TO	0.02	TO	0.34	0.22	TO	GOpt	2875.17
wtsM2_06	GOpt	228.18	15.52	0.01	TO	0.01	TO	0.33	0.02	TO	GOpt	1957.59
wtsM2_07	GOpt	759.96	16.15	0.26	TO	0.30	TO	0.22	0.04	TO	1.59	TO
wtsM2_08	388.62	TO	21.54	14.37	TO	14.72	TO	0.90	20.08	TO	19.94	TO
wtsM2_09	Inf	TO	42.28	61.99	TO	64.53	TO	13.96	56.72	TO	55.85	TO
wtsM2_10	76.48	TO	15.73	0.10	TO	0.07	TO	0.32	0.22	TO	0.22	TO

Table 3 continued

Instances	BARON		BT	AMP-cpx		AMP-grb		PBT	AMP-cpx		AMP-grb	
	Gap	T	T^+	Gap	T	Gap	T	T^+	Gap	T	Gap	T
wtsM2_11	107.56	TO	22.64	9.76	TO	3.74	TO	1.09	13.81	TO	9.18	TO
wtsM2_12	85.35	TO	39.10	11.90	TO	11.92	TO	3.44	6.03	TO	11.02	TO
wtsM2_13	54.04	TO	111.39	2.01	TO	3.95	TO	17.00	2.17	TO	2.10	TO
wtsM2_14	46.24	TO	19.09	6.64	TO	4.71	TO	1.10	1.93	TO	1.93	TO
wtsM2_15	Inf	TO	14.93	0.29	TO	0.50	TO	0.34	0.51	TO	0.48	TO
wtsM2_16	47.77	TO	22.54	8.76	TO	6.91	TO	1.18	9.40	TO	5.11	TO
lee1	GOpt	145.55	13.28	GOpt	12.50	GOpt	13.55	0.22	10.00	TO	0.01	TO
lee2	GOpt	590.08	14.92	0.58	TO	0.37	TO	5.35	0.43	TO	0.07	TO
meyer4	80.40	TO	15.78	GOpt	4.22	GOpt	4.47	238.47	GOpt	14.76	GOpt	13.68
meyer10	239.70	TO	44.68	9.74	TO	GOpt	2925.36	63.79	GOpt	TO	GOpt	1189.71
meyer15	2556.37	TO	3877.09	3.44	TO	0.08	TO	17868.96	GOpt	TO	GOpt	TO

Here, we compare the run times of BARON, BT-AMP with $\Delta = 8$ and PBT-AMP with $\Delta = 10$. CPLEX and Gurobi are the underlying solvers for AMP. Values under "Gap" and "T, T^+" are in % and seconds, respectively. "Inf" implies that the solver failed to provide a bound within the prescribed time limit
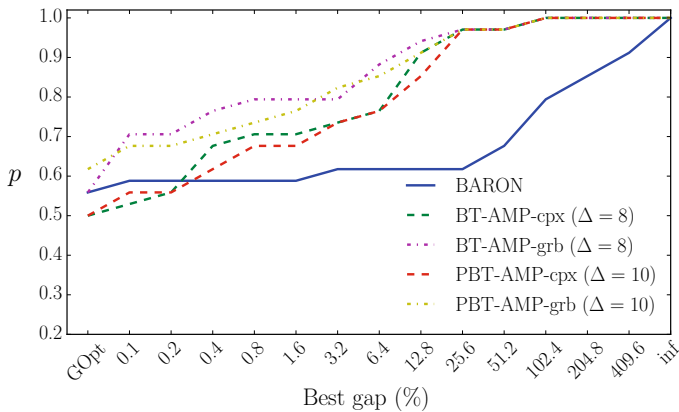
(a) Comparison of best gap
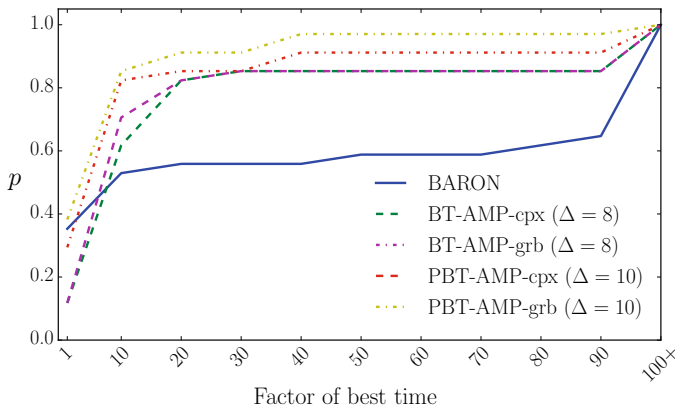
(b) Comparison of best run times

Fig. 9 Performance profiles of AMP (with OBBT) and BARON. In **a** the x axis plots the optimality gap of the algorithms and the y axis plots fraction of instances. Plot **a** tracks the number of instances where an algorithm is able achieve the specified optimality gap. In **b** the x axis denotes the run time ratio of an algorithm with the best run time of any algorithm. The y axis denotes the fraction of instances. Plot **b** tracks the number of times an algorithm's run time is within a specified factor of the best run time of any algorithm. In both figures, higher is better. Overall, AMP performs better than BARON on p proportion of instances within a factor of the best gap and with the best run times

the performance improvement starts at 0.2%. However, there is an increase in run times due to bound-tightening (Fig. 9b), that allows to AMP to achieve this improvement.

Remark 2 *meyer15*, a generalized pooling problem-based instance, is classified as a large-scale MINLP and is very hard for global optimization. The current best known gap for this instance is 0.1% [9,36]. PBT-AMP with Gurobi has closed this problem by proving the *global optimum* for the first time (*943,734.0215*–Table 3).

Table 4 Summary of the performance of AMP with and without OBBT on all instances

Instances	BARON		BT-AMP-grb				PBT-AMP-grb			
	Gap	T	Δ^*	Gap	T^+	T	Δ^*	Gap	T^+	T
fuel	**GOpt**	**0.03**	8	GOpt	0.01	0.04	8	GOpt	0.01	0.04
ex1223a	**GOpt**	**0.02**	8	GOpt	19.71	0.01	32	GOpt	0.31	0.01
ex1264	GOpt	1.44	4	GOpt	12.47	0.59	4	**GOpt**	**0.84**	**0.49**
ex1265	GOpt	13.30	16	GOpt	6.02	0.45	16	**GOpt**	**0.92**	**0.46**
ex1266	GOpt	10.81	8	GOpt	13.75	0.06	4	**GOpt**	**1.18**	**0.09**
eniplac	GOpt	207.37	32	GOpt	16.04	0.45	32	**GOpt**	**3.34**	**0.64**
util	**GOpt**	**0.10**	4	GOpt	9.92	0.08	4	GOpt	0.56	0.09
meanvarx	**GOpt**	**0.05**	16	GOpt	20.69	21.17	16	GOpt	3.45	15.57
blend029	GOpt	2.46	32	GOpt	15.05	0.92	16	**GOpt**	**1.09**	**1.33**
blend531	GOpt	111.79	8	**GOpt**	**44.67**	**22.56**	32	GOpt	239.17	84.97
blend146	2.20	TO	8	**0.10**	**30.95**	**TO**	8	2.10	23.06	TO
blend718	175.10	TO	32	**GOpt**	**28.76**	**889.28**	8	GOpt	21.80	1101.56
blend480	GOpt	326.95	8	**GOpt**	**137.62**	**108.93**	16	GOpt	948.27	2185.03
blend721	GOpt	548.90	16	GOpt	29.44	92.27	32	**GOpt**	**9.87**	**90.91**
blend852	0.08	TO	16	GOpt	41.73	323.86	16	**GOpt**	**14.16**	**323.31**

Table 4 continued

Instances	BARON		BT-AMP-grb				PBT-AMP-grb			
	Gap	T	Δ^*	Gap	T^+	T	Δ^*	Gap	T^+	T
wtsM2_05	**GOpt**	**153.30**	16	GOpt	14.82	2482.73	16	GOpt	0.33	2483.36
wtsM2_06	**GOpt**	**228.18**	16	GOpt	15.52	2058.92	16	GOpt	0.39	2057.20
wtsM2_07	**GOpt**	**759.96**	8	0.30	16.15	TO	8	0.30	0.24	TO
wtsM2_08	388.62	TO	4	**8.25**	**21.54**	**TO**	4	8.25	24.43	TO
wtsM2_09	Inf	TO	16	44.57	42.80	TO	16	**43.66**	**288.88**	**TO**
wtsM2_10	76.48	TO	8	0.07	15.73	TO	8	**0.06**	**0.39**	**TO**
wtsM2_11	107.56	TO	8	3.74	22.64	TO	8	**2.39**	**1.11**	**TO**
wtsM2_12	85.35	TO	4	(6.89) 7.29	39.10	TO	8	(6.95) **7.34**	**3.70**	**TO**
wtsM2_13	54.04	TO	8	**3.95**	**111.39**	**TO**	4	6.92	130.31	TO
wtsM2_14	46.24	TO	32	2.71	19.09	TO	8	**2.25**	**0.67**	**TO**
wtsM2_15	Inf	TO	32	**0.20**	**14.93**	**TO**	16	0.28	0.35	TO
wtsM2_16	47.77	TO	4	(2.61) **5.81**	**22.54**	**TO**	4	(3.04) 6.24	24.41	TO
lee1	GOpt	145.55	8	GOpt	13.28	13.55	8	**GOpt**	**0.27**	**13.68**
lee2	**GOpt**	**590.08**	16	0.36	14.92	TO	4	0.38	2.96	TO
meyer4	80.40	TO	8	**GOpt**	**15.78**	**4.47**	4	GOpt	77.61	8.45
meyer10	239.70	TO	4	**GOpt**	**44.68**	**760.36**	4	GOpt	34.67	775.74
meyer15	2556.37	TO	4	0.02	3877.09	TO	4	**GOpt**	19218.84	**538.56**

Here, we compare the run times of BARON, BT-AMP and PBT-AMP with tuned values of Δ for each instance. Values under "Gap" and "T, T^+" are in % and seconds, respectively. "Inf" implies that the solver failed to provide a bound within the prescribed time limit. Gap values shown within parenthesis are evaluated using global optimum values instead of the best-found upper bound by AMP. For each instance, the bold face font represents best run time or the best optimality gap (if the solve times out)

4.3.2 Tuned Δ parameter

In Table 4, we show the results of AMP (with OBBT) when Δ is tuned for each problem instance and these results are compared with BARON. For the purposes of this article, Δ is tuned by running AMP with $\Delta = \{4, 8, 10, 16, 32\}$ on each instance. We then choose the value of Δ that provides the best lower bound (global optimal in many cases) in the minimum amount of computation time. This tuned value of Δ is denoted by Δ^* in Table 4. Developing adaptive and automatic tuning heuristic-algorithms for computing the value of Δ remains an open question and is a subject of future work. As the performance of AMP is consistently the strongest with Gurobi, we present those results. Overall, this table shows the best results for AMP. Column two of this table shows the run times of BARON. Similar to Table 3, the time limit for BARON is the sum of 3600 s and the maximum of the run times of the BT and PBT algorithms (no time limit on BT and PBT). Column three tabulates the performance of BT-AMP with Gurobi by choosing the best Δ parameter for each instance. Overall, BT-AMP and PBT-AMP performed better than BARON on 26 out of 32 instances. As discussed in detail in Sect. 4.3.1, similar observations about the performance of our algorithms also hold for this table. Again, the performance of PBT, despite the use of discrete optimization, indicates that PBT is the strongest bound-tightening procedure. On the large MINLP instance *meyer15*, PBT has large computational overhead, but this overhead pays off when AMP converges to the global optimum in 538.56 s. As noted in the earlier remark, this was an open instance prior to this work.

Table 3 is summarized with the cumulative distribution plot shown in Fig. 10. Figure 10a indicates that AMP, BT-AMP and PBT-AMP with Gurobi are better than BARON in finding the best lower bounds. Though the proportion of instances for which global optima are attained is not significantly different than BARON, the proportion of instances for which better lower bounds are found using AMP-based algorithms is larger. Also, the advantages of BT and PBT-based bound-tightening in AMP is evident from the fact that the proportion of instances that find global optimum is higher. As expected, Figure 10b suggests that AMP with Gurobi is overall faster than BARON on the easiest instances, but on harder instances this speed is tempered by a degradation in solution quality.

4.4 Sensitivity of MINLP structure

In Fig. 11, we classify the MINLP instances to understand how problem structure influences the success of AMP. There are various possible classification measures and we use the total number of variables that are part of multilinear terms. This is because our algorithm heavily depends on multi-variate partitioning on the nonlinear terms. Thus, it is likely that a measure like this influences the performance of AMP. Consider the following simple example that describes the measure clearly: Let x_i, $\forall i = 1, \ldots, n$ be the variables in a problem with a linear objective and one nonlinear constraint, $\left(\prod_{i=1}^{k} x_i + \prod_{i=2}^{k+1} x_i \right) \geqslant M$, such that $2 \leqslant k \leqslant n-1$. Then, the number of variables in mutlilinear terms is $k + 1$.

It is clear from the figure that both AMP and BT-AMP performs very well on instances that have large numbers of variables ($\geqslant 25$) in the multilinear terms. We also observe that while executing OBBT incurs a computational overhead (ratio up to ≈ 16), there are many instances below the unit ratio value (blue dashed line). Overall, these plots support the observation that increasing the number of variables in multilinear terms are indicator of success when executing AMP.
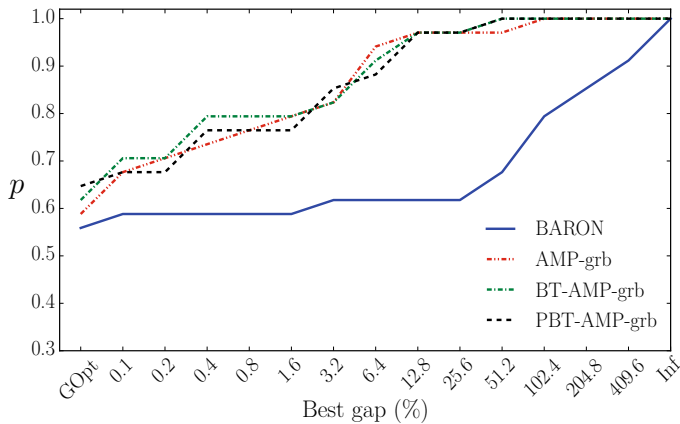
(a) Comparison on Best Gap %
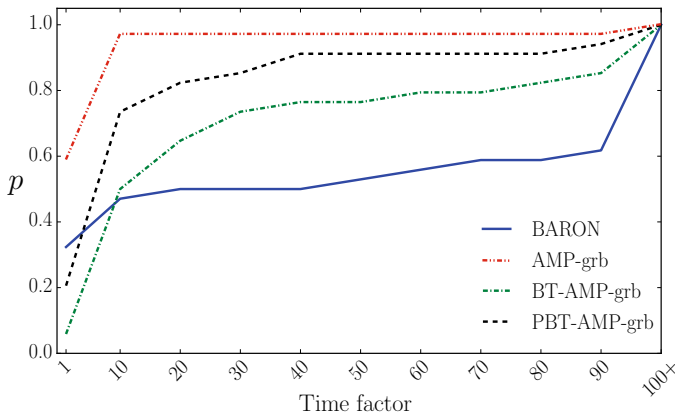
(b) Comparison on Best CPU Times

Fig. 10 Performance profiles of AMP (with OBBT and tuned Δ) and BARON. In **a** the x axis plots the optimality gap of the algorithms and the y axis plots fraction of instances. Plot **a** tracks the number of instances where an algorithm is able achieve the specified optimality gap. In **b** the x axis denotes the run time ratio of an algorithm with the best run time of any algorithm. The y axis denotes the fraction of instances. Plot **b** tracks the number of times an algorithm's run time is within a specified factor of the best run time of any algorithm. In both figures, higher is better. Overall, AMP-based algorithms perform better than BARON on p proportion of instances within a factor of the best gap and with the best run times

4.5 OBBT results for *meyer15*

One of the primary observations made in this paper is the importance of MIP-based sequential OBBT on medium-scale MINLPs. However, for a given large-scale MINLP, one of the drawbacks of BT and PBT is that it solves MILPs to tighten the variable bounds. Solving MILPs can be time consuming, in particular on instances like *meyer15*.[6] Here, we focus on the run time issues associated with solving MILPs, suggest approaches for managing that run time and still get much of their benefits in bound-tightening.

---

[6] *meyer15* is a generalized pooling problem instance. These problems are typically considered hard (bilinear) MINLP for global optimization [9,36].
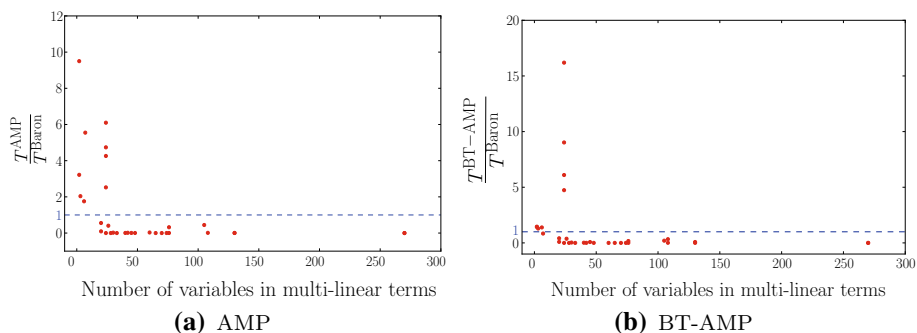
Fig. 11 Illustration of the ratio of run times of AMP and BT-AMP algorithms (with tuned parameters) to BARON. The y axis denotes the ratio and the x axis denotes the total number of variables in mutlilinear terms in a given MINLP instance. The blue dashed line indicates a ratio of 1. All red points correspond to a single instance. A point below the blue line indicates a ratio in favor of AMP. (Color figure online)

Table 5 BT and PBT run times on *meyer15* with and without a limit on the run time of every MILP solved during the bound-tightening phase

	BT	PBT				
		$\Delta = 4$	$\Delta = 8$	$\Delta = 10$	$\Delta = 16$	$\Delta = 32$
Without time limit	3877	19218	32130	29586	11705	17868
With time limit	3830	3914	4059	4014	3635	3621

Here, a 10-s time limit was used. The imposition of this limit reduces the total run time of PBT to close to BT

Table 6 AMP run times on *meyer15* instance preceded by bound-tightening with (BT-lim, PBT-lim) and without limit (BT, PBT) on run time per iteration

	$\Delta = 4$		$\Delta = 8$		$\Delta = 10$		$\Delta = 16$		$\Delta = 32$	
	Gap	T	Gap	T	Gap	T	Gap	T	Gap	T
After BT	0.02	TO	0.08	TO	0.12	TO	0.31	TO	0.15	TO
After BT-lim	0.37	TO	0.14	TO	0.17	TO	0.69	TO	0.90	TO
After PBT	GOpt	536.56	GOpt	1600	GOpt	TO	0.20	TO	0.15	TO
After PBT-lim	0.37	TO	0.90	TO	0.19	TO	0.70	TO	0.90	TO

Values under "Gap" and "T" are in % and seconds, respectively. Bold font represents the best result in each row

Table 5 summarizes the run times of BT and PBT on *meyer15* for various values of Δ. As shown in the first row of this table, the run time varies drastically when the MILP is solved to optimality in every iteration of BT and PBT. To reduce this run time, we imposed a time limit on every min- and max-MILP of *10 s*. Since early termination of MILP does not guarantee optimal primal-feasible solutions, incumbent solutions are not valid for bound-tightening. Instead, we use the best lower bound maintained by the solver. This ensures the validity of the tightened bounds. As shown in the second row of Table 5, the run times of PBT are drastically reduced.

Table 6 summarizes the results of AMP based on the tightened variable bounds presented in Table 5. Interestingly, on *meyer15*, we observed that AMP converges to near optimal solutions (sometimes, even better than solving full MILPs) when the time limit on MILP solvers is

Table 7 Overall structure of the MINLP problems. The first column describes the name of the problem instance. The second column cites the source of the problem. The third column shows the optimal solution. The fourth, fifth and sixth columns show the number of constraints, binary variables, and continuous variables, respectively. The seventh column indicates the partitioned continuous variables. "ALL" refers to all variables in mutlilinear terms and "VC" refers to variables in the minimum vertex cover as described in [8]. The final column shows the number of mutlilinear terms

Instance	References	GOpt	#Cons	#BVars	#CVars	#CVars-P	#ML
NLP1	[38]	7049.248	14	0	8	ALL	5
fuel	[9]	8566.119	15	3	12	VC	3
ex1223a	[9]	4.580	9	4	3	VC	3
ex1264	[9]	8.6	55	68	20	VC	16
ex1265	[9]	10.3	74	100	30	VC	25
ex1266	[9]	16.3	95	138	42	VC	36
eniplac	[9]	− 132117.083	189	24	117	VC	66
util	[9]	999.578	167	28	117	ALL	5
meanvarx	[9]	14.369	44	14	21	VC	28
blend029	[9]	13.359	213	36	66	VC	28
blend531	[9]	20.039	736	104	168	VC	146
blend146	[9]	45.297	624	87	135	VC	104
blend718	[9]	7.394	606	87	135	VC	100
blend480	[9]	9.227	884	124	188	VC	152
blend721	[9]	13.5268	627	87	135	VC	104
blend852	[9]	53.9626	2412	120	184	VC	152
wtsM2_05	[35]	229.7008	152	0	134	VC	48
wtsM2_06	[35]	173.4784	152	0	134	VC	48
wtsM2_07	[35]	80.77892	152	0	134	VC	48
wtsM2_08	[35]	109.4014	335	0	279	VC	84
wtsM2_09	[35]	124.4421	573	0	517	ALL	210
wtsM2_10	[35]	586.68	138	0	156	VC	60
wtsM2_11	[35]	2127.115	252	0	304	VC	112
wtsM2_12	[35]	1201.038	408	0	517	VC	220
wtsM2_13	[35]	1564.958	783	0	1040	VC	480
wtsM2_14	[35]	513.009	205	0	209	VC	90
wtsM2_15	[35]	2446.429	152	0	134	VC	48
wtsM2_16	[35]	1358.663	234	0	244	VC	126
lee1	[34]	− 4640.0824	82	9	40	VC	24
lee2	[34]	− 3849.2654	92	9	44	VC	36
meyer4	[34]	1086187.137	118	55	63	VC	48
meyer10	[34]	1086187.137	423	187	207	VC	300
meyer15	[34]	943734.0215	768	352	382	VC	675

imposed. This is an important feature for tuning the time spent tightening bounds versus bound solution quality. Further, this intriguing result suggests further study of MIP-based relaxations for OBBT of MINLPs, which we delegate for future work (Table 7).

5 Conclusions

In this work, we developed an approach for adaptively partitioning nonconvex functions in MINLPs. We show that an adaptive partitioning of the domains of variables outperforms uniform partitioning, though the latter exhibits better optimality gaps in the first few iterations of the lower-bounding algorithm. We also show that bound-tightening techniques can be applied in conjunction with adaptive partitioning to improve convergence dramatically. We then use combinations of these techniques to develop an algorithm for solving MINLPs to global optimality. Our numerical experiments on MINLPs with polynomials suggests that this is a very strong approach with an advantage of having very few tuning parameters in contrast to the existing methods.

We have seen that using a well-designed MIP-based method with adaptive partitioning schemes is an attractive way of tackling MINLPs. With an apriori fixed tolerance, we get global optimum solutions by utilizing the well-developed state-of-the-art MIP solvers. However, though AMP is relatively faster than the available global solvers, we observed that the computation times remain large for MINLPs with large number of nonconvex terms, thus motivating a multitude of directions for further developments. First, it will be important to consider existing classical nonlinear programming techniques, such as dual-based bound-contraction, partition elimination within the branch-and-bound search tree, bound-tightening at sub nodes [37], and constraint propagation methods [3], which can tremendously speed-up our algorithm. Second, providing apriori guarantees on the size of the added partitions (Δ) that leads to faster tightening of the relaxations. This will support automatic tuning of Δ from within AMP. Third, recent developments on generating tight convex hull-reformulation-based cutting planes for solving convex generalized disjunctive programs will be very effective for attaining faster convergence to global optimum [51]. Finally, extensions of our methods from polynomial to general nonconvex functions (including fractional exponents, transcendental functions and disjunctions of nonconvex functions) will be another direction that will have relevance to numerous practical applications.

A: Appendix

A.1: Sensitivity analysis of Δ

One of the important details of MINLP algorithms and approaches is their parameterization. As seen in the earlier sections, AMP is no different. The quality of the solutions depend heavily on the choice of Δ. However, in spite of this problem specific dependence, it is often interesting to identify reasonable default values. Table 8 presents computational results on all instances for different choices of Δ. From these results, AMP is most effective when Δ is between 4 and 10.

Table 8 This table shows a sensitivity analysis of AMP's performance to the choice of Δ

Instances	$\Delta \leq 4$		$4 < \Delta \leq 10$		$\Delta > 10$	
	Gap(%)	T	Gap(%)	T	Gap(%)	T
p1	GOpt	0.74	GOpt	0.24	**GOpt**	**0.06**
p2	GOpt	0.60	GOpt	0.20	**GOpt**	**0.10**
fuel	**GOpt**	**0.05**	GOpt	0.06	GOpt	0.07
ex1223a	GOpt	0.03	**GOpt**	**0.02**	GOpt	0.02
ex1264	GOpt	1.92	**GOpt**	**0.79**	GOpt	1.03
ex1265	GOpt	2.24	**GOpt**	**0.28**	GOpt	0.80
ex1266	GOpt	0.20	GOpt	0.23	**GOpt**	**0.16**
eniplac	GOpt	2.39	GOpt	1.46	**GOpt**	**0.75**
util	GOpt	2.52	GOpt	2.14	**GOpt**	**0.55**
meanvarx	GOpt	967.70	GOpt	118.80	**GOpt**	**70.09**
blend029	GOpt	1.98	GOpt	1.33	**GOpt**	**1.00**
blend531	GOpt	88.60	GOpt	74.33	**GOpt**	**49.76**
blend146	23.34	TO	**1.60**	**TO**	3.20	TO
blend718	GOpt	1263.41	**GOpt**	**581.68**	GOpt	889.72
blend480	0.10	TO	**0.02**	**TO**	0.02	TO
blend721	GOpt	486.17	**GOpt**	**44.13**	GOpt	176.11
blend852	0.01	TO	**GOpt**	**144.26**	GOpt	322.80
wtsM2_05	GOpt	2236.45	GOpt	2545.41	**GOpt**	**386.95**
wtsM2_06	0.02	TO	**GOpt**	**519.38**	GOpt	972.20
wtsM2_07	0.77	TO	0.57	TO	**0.54**	**TO**
wtsM2_08	**7.92**	**TO**	9.28	TO	11.96	TO
wtsM2_09	**7.47**	**TO**	68.58	TO	68.58	TO
wtsM2_10	0.11	TO	0.11	TO	**0.10**	**TO**
wtsM2_11	**6.10**	**TO**	6.27	TO	10.41	TO
wtsM2_12	6.49	TO	8.69	TO	**4.00**	**TO**
wtsM2_13	7.37	TO	**2.03**	**TO**	10.27	TO
wtsM2_14	4.06	TO	5.59	TO	**1.43**	**TO**
wtsM2_15	**0.17**	**TO**	0.17	TO	0.57	TO
wtsM2_16	5.73	TO	8.17	TO	**5.25**	**TO**
lee1	GOpt	73.19	**GOpt**	**13.61**	0.03	TO
lee2	0.38	TO	**0.02**	**TO**	0.08	TO
meyer4	GOpt	64.82	**GOpt**	**5.33**	GOpt	20.74
meyer10	GOpt	684.63	**GOpt**	**133.47**	9.70	TO
meyer15	**0.10**	**TO**	0.33	TO	0.15	TO
Summary	7		14		14	

Here, we bin results by $\Delta \leq 4$, Δ between 4 and 10, and $\Delta > 10$. From these results, it is clear that most of the good choices of Δ are between 4 and 10 and this is our recommended choice for this parameter. For each instance, the bold face font represents best run time or the best optimality gap (if the solve times out)

A.2: Logarithmic and linear encodings of partition variables

In Sect. 2, the discussion on piecewise convex relaxations described formulations that encoded the partition variables with a linear number of variables and a logarithmic number of variables

Table 9 This table compares the logarithmic formulation of partition variables with the linear representation

Instances	$F(\Delta = 4)$	$F(\Delta = 8)$	$F(\Delta = 10)$	$F(\Delta = 16)$	$F(\Delta = 32)$	Total
eniplac	lin	lin	log	log	log	3
blend531	lin	lin	log	lin	lin	1
blend146	lin	lin	log	lin	lin	1
blend718	lin	lin	lin	lin	log	1
blend480	lin	lin	log	lin	lin	1
blend721	lin	lin	lin	lin	lin	0
blend852	log	lin	log	lin	log	3
wtsM2_05	lin	lin	lin	lin	lin	0
wtsM2_06	lin	lin	lin	lin	lin	0
wtsM2_07	lin	lin	lin	lin	lin	0
wtsM2_08	log	lin	log	lin	log	3
wtsM2_09	lin	log	log	log	log	4
wtsM2_10	lin	lin	lin	lin	lin	0
wtsM2_11	lin	lin	lin	lin	lin	0
wtsM2_12	lin	lin	lin	lin	lin	0
wtsM2_13	lin	lin	lin	lin	lin	0
wtsM2_14	lin	lin	lin	lin	lin	0
wtsM2_15	log	lin	lin	log	lin	2
wtsM2_16	lin	lin	lin	lin	lin	0
lee1	log	log	log	log	log	5
lee2	log	lin	lin	lin	lin	1
meyer4	log	log	lin	lin	log	3
meyer10	lin	lin	log	log	lin	2
meyer15	log	log	lin	lin	log	3
Total	7	4	9	5	8	33

Each column indicates the formulation with the fastest runtime for different choices of Δ. The last column enumerates the number of times the logarithmic formulation is better

[52]. Table 9 compares the performance of AMP using both formulations. Despite fewer variables in the logarithmic formulation, this encoding is only effective on a few problems, generally on problems that require a significant number of partitions. These results suggest that when the logarithmic encoding has nearly the same number of partition variables as the linear encoding, the linear encoding is more effective.

References

1. Achterberg, T.: SCIP: solving constraint integer programs. Math. Progr. Comput. **1**(1), 1–41 (2009)
2. Al-Khayyal, F.A., Falk, J.E.: Jointly constrained biconvex programming. Math. Oper. Res. **8**(2), 273–286 (1983)
3. Belotti, P.: Bound reduction using pairs of linear inequalities. J. Glob. Optim. **56**(3), 787–819 (2013)
4. Belotti, P., Cafieri, S., Lee, J., Liberti, L.: On feasibility based bounds tightening (2012). https://hal.archives-ouvertes.fr/file/index/docid/935464/filename/377.pdf
5. Belotti, P., Lee, J., Liberti, L., Margot, F., Wächter, A.: Branching and bounds tightening techniques for non-convex MINLP. Optim. Methods Softw. **24**(4–5), 597–634 (2009)

6. Bent, R., Nagarajan, H., Sundar, K., Wang, S., Hijazi, H.: A polyhedral outer-approximation, dynamic-discretization optimization solver, 1.x. Tech. rep., Los Alamos National Laboratory, Los Alamos, NM, USA (2017). https://github.com/lanl-ansi/POD.jl

7. Bergamini, M.L., Grossmann, I., Scenna, N., Aguirre, P.: An improved piecewise outer-approximation algorithm for the global optimization of MINLP models involving concave and bilinear terms. Comput. Chem. Eng. **32**(3), 477–493 (2008)

8. Boukouvala, F., Misener, R., Floudas, C.A.: Global optimization advances in mixed-integer nonlinear programming, MINLP, and constrained derivative-free optimization, CDFO. Eur. J. Oper. Res. **252**(3), 701–727 (2016)

9. Bussieck, M.R., Drud, A.S., Meeraus, A.: MINLPLib–a collection of test models for mixed-integer nonlinear programming. Inf. J. Comput. **15**(1), 114–119 (2003)

10. Cafieri, S., Lee, J., Liberti, L.: On convex relaxations of quadrilinear terms. J. Glob. Optim. **47**(4), 661–685 (2010)

11. Castro, P.M.: Normalized multiparametric disaggregation: An efficient relaxation for mixed-integer bilinear problems. J. Glob. Optim. **64**(4), 765–784 (2016)

12. Castro, P.M.: Tightening piecewise McCormick relaxations for bilinear problems. Comput. Chem. Eng. **72**, 300–311 (2015)

13. Coffrin, C., Hijazi, H.L., Van Hentenryck, P.: Strengthening convex relaxations with bound tightening for power network optimization. In: Principles and Practice of Constraint Programming, pp. 39–57. Springer, Berlin (2015)

14. Dunning, I., Huchette, J., Lubin, M.: JuMP: A modeling language for mathematical optimization. SIAM Rev. **59**(2), 295–320 (2017)

15. D'Ambrosio, C., Lodi, A., Martello, S.: Piecewise linear approximation of functions of two variables in milp models. Oper. Res. Lett. **38**(1), 39–46 (2010)

16. Faria, D.C., Bagajewicz, M.J.: Novel bound contraction procedure for global optimization of bilinear MINLP problems with applications to water management problems. Comput. Chem. Eng. **35**(3), 446–455 (2011)

17. Faria, D.C., Bagajewicz, M.J.: A new approach for global optimization of a class of MINLP problems with applications to water management and pooling problems. AIChE J. **58**(8), 2320–2335 (2012)

18. Grossmann, I.E., Trespalacios, F.: Systematic modeling of discrete-continuous optimization models through generalized disjunctive programming. AIChE J. **59**(9), 3276–3295 (2013)

19. Hasan, M., Karimi, I.: Piecewise linear relaxation of bilinear programs using bivariate partitioning. AIChE J. **56**(7), 1880–1893 (2010)

20. Hijazi, H., Coffrin, C., Van Hentenryck, P.: Convex quadratic relaxations for mixed-integer nonlinear programs in power systems. Math. Progr. Comput. **9**(3), 321–367 (2017)

21. Hock, W., Schittkowski, K.: Test examples for nonlinear programming codes. J. Optim. Theory Appl. **30**(1), 127–129 (1980)

22. Horst, R., Pardalos, P.M.: Handbook of global optimization, vol. 2. Springer, Berlin (2013)

23. Horst, R., Tuy, H.: Global optimization: deterministic approaches. Springer, Berlin (2013)

24. Karuppiah, R., Grossmann, I.E.: Global optimization for the synthesis of integrated water systems in chemical processes. Comput. Chem. Eng. **30**(4), 650–673 (2006)

25. Kocuk, B., Dey, S.S., Sun, X.A.: Strong SOCP relaxations for the optimal power flow problem. Oper. Res. **64**(6), 1177–1196 (2016)

26. Kolodziej, S.P., Grossmann, I.E., Furman, K.C., Sawaya, N.W.: A discretization-based approach for the optimization of the multiperiod blend scheduling problem. Comput. Chem. Eng. **53**, 122–142 (2013)

27. Li, H.L., Huang, Y.H., Fang, S.C.: A logarithmic method for reducing binary variables and inequality constraints in solving task assignment problems. Inf. J. Comput. **25**(4), 643–653 (2012)

28. Liberti, L., Lavor, C., Maculan, N.: A branch-and-prune algorithm for the molecular distance geometry problem. Int. Trans. Oper. Res. **15**(1), 1–17 (2008)

29. Lu, M., Nagarajan, H., Bent, R., Eksioglu, S., Mason, S.: Tight piecewise convex relaxations for global optimization of optimal power flow. In: Power Systems Computation Conference (PSCC), pp. 1–7. IEEE (2018)

30. Lu, M., Nagarajan, H., Yamangil, E., Bent, R., Backhaus, S., Barnes, A.: Optimal transmission line switching under geomagnetic disturbances. IEEE Trans. Power Syst. **33**(3), 2539–2550 (2018). https://doi.org/10.1109/TPWRS.2017.2761178

31. Luedtke, J., Namazifar, M., Linderoth, J.: Some results on the strength of relaxations of multilinear functions. Math. Progr. **136**(2), 325–351 (2012)

32. McCormick, G.P.: Computability of global solutions to factorable nonconvex programs: Part i–convex underestimating problems. Math. Progr. **10**(1), 147–175 (1976)

33. Meyer, C.A., Floudas, C.A.: Global optimization of a combinatorially complex generalized pooling problem. AIChE J. **52**(3), 1027–1037 (2006)
34. Misener, R., Floudas, C.: Generalized pooling problem (2011). Available from Cyber-Infrastructure for MINLP [www.minlp.org, a collaboration of Carnegie Mellon University and IBM Research] at: www.minlp.org/library/problem/index.php?i=123
35. Misener, R., Floudas, C.A.: GloMIQO: Global mixed-integer quadratic optimizer. J. Glob. Optim. **57**(1), 3–50 (2013)
36. Misener, R., Thompson, J.P., Floudas, C.A.: APOGEE: Global optimization of standard, generalized, and extended pooling problems via linear and logarithmic partitioning schemes. Comput. Chem. Eng. **35**(5), 876–892 (2011)
37. Mouret, S., Grossmann, I.E., Pestiaux, P.: Tightening the linear relaxation of a mixed integer nonlinear program using constraint programming. In: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, pp. 208–222. Springer, Berlin (2009)
38. Nagarajan, H., Lu, M., Yamangil, E., Bent, R.: Tightening McCormick relaxations for nonlinear programs via dynamic multivariate partitioning. In: International Conference on Principles and Practice of Constraint Programming, pp. 369–387. Springer, Berlin (2016)
39. Nagarajan, H., Pagilla, P., Darbha, S., Bent, R., Khargonekar, P.: Optimal configurations to minimize disturbance propagation in manufacturing networks. In: American Control Conference (ACC), 2017, pp. 2213–2218. IEEE (2017)
40. Nagarajan, H., Sundar, K., Hijazi, H., Bent, R.: Convex hull formulations for mixed-integer multilinear functions. In: Proceedings of the XIV International Global Optimization Workshop (LEGO 18) (2018)
41. Nagarajan, H., Yamangil, E., Bent, R., Van Hentenryck, P., Backhaus, S.: Optimal resilient transmission grid design. In: Power Systems Computation Conference (PSCC), 2016, pp. 1–7. IEEE (2016)
42. Puranik, Y., Sahinidis, N.V.: Domain reduction techniques for global NLP and MINLP optimization. Constraints **22**(3), 338–376 (2017)
43. Rikun, A.D.: A convex envelope formula for multilinear functions. J. Glob. Optim. **10**, 425–437 (1997). https://doi.org/10.1023/A:1008217604285
44. Ruiz, J.P., Grossmann, I.E.: Global optimization of non-convex generalized disjunctive programs: a review on reformulations and relaxation techniques. J. Glob. Optim. **67**(1–2), 43–58 (2017)
45. Ryoo, H.S., Sahinidis, N.V.: Global optimization of nonconvex nlps and MINLPs with applications in process design. Comput. Chem. Eng. **19**(5), 551–566 (1995)
46. Ryoo, H.S., Sahinidis, N.V.: Analysis of bounds for multilinear functions. J. Glob. Optim. **19**(4), 403–424 (2001)
47. Sahinidis, N.V.: Baron: A general purpose global optimization software package. J. Glob. Optim. **8**(2), 201–205 (1996)
48. Speakman, E.E.: Volumetric Guidance for Handling Triple Products in Spatial Branch-and-Bound by. Ph.D. thesis, University of Michigan (2017)
49. Tawarmalani, M., Sahinidis, N.V.: A polyhedral branch-and-cut approach to global optimization. Math. Program. **103**(2), 225–249 (2005)
50. Teles, J.P., Castro, P.M., Matos, H.A.: Univariate parameterization for global optimization of mixed-integer polynomial problems. Eur. J. Oper. Res. **229**(3), 613–625 (2013)
51. Trespalacios, F., Grossmann, I.E.: Cutting plane algorithm for convex generalized disjunctive programs. Inf. J. Comput. **28**(2), 209–222 (2016)
52. Vielma, J.P., Nemhauser, G.L.: Modeling disjunctive constraints with a logarithmic number of binary variables and constraints. Math. Progr. **128**(1), 49–72 (2011)
53. Wicaksono, D.S., Karimi, I.: Piecewise MILP under-and overestimators for global optimization of bilinear programs. AIChE J. **54**(4), 991–1008 (2008)
54. Wu, F., Nagarajan, H., Zlotnik, A., Sioshansi, R., Rudkevich, A.: Adaptive convex relaxations for gas pipeline network optimization. In: American Control Conference (ACC), 2017, pp. 4710–4716. IEEE (2017)