Decomposing Optimization-Based Bounds Tightening Problems Via Graph Partitioning

Michael Bynum[1], Anya Castillo[2], Bernard Knueven[3], Carl Laird[1], John Siirola[1], and Jean-Paul Watson[4]

[1] Discrete Math & Optimization, Sandia National Laboratories, Albuquerque, NM 87185, Email: mlbynum@sandia.gov, cdlaird@sandia.gov, jdsiiro@sandia.gov

[2] Data Science & Applications, Sandia National Laboratories, Albuquerque, NM 87185, Email: anya.castillo@gmail.com

[3] Computational Science Center, National Renewable Energy Laboratory, Golden, CO 80401, Email: Bernard.Knueven@nrel.gov

[4] Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA 94550 , Email: jeanpaulwatson@llnl.gov

Abstract

Bounds tightening or domain reduction is a critical refinement technique used in global optimization algorithms for nonlinear and mixed-integer nonlinear programming problems. Bounds tightening can strengthen convex relaxations and reduce the size of branch and bounds trees. An effective but computationally intensive bounds tightening technique is optimization-based bounds tightening (OBBT). In OBBT, each variable is typically minimized and maximized subject to a convex relaxation of the original problem in order to obtain tighter variable bounds. In this paper, we present two variants of a scalable bounds tightening algorithm that decomposes the majority of the bounds tightening problems into much smaller problems via graph partitioning. Numerical results on a set of optimal power flow test problems and problems from MINLPLib demonstrate that our proposed algorithms can be nearly as effective as traditional OBBT in terms of domain reduction. Furthermore, the algorithms are significantly more computationally efficient and scale much better with problem size. For large problems, our decomposition algorithm can be over an order of magnitude faster than traditional OBBT and nearly as effective.

1 Introduction

Mixed-integer nonlinear programming (MINLP) addresses a wide range of important optimization problems [31]. Most deterministic global optimization algorithms for solving MINLPs utilize either branch-and-bound (B&B) or multitree methods (e.g., outer-approximation) [5, 13, 24]. MINLP B&B algorithms use the solution of convex relaxations (usually linear relaxations) of the original problem to obtain objective lower bounds and local nonlinear

programming (NLP) solutions to obtain objective upper bounds; we generally assume a minimization objective. The objective bounds are progressively refined with spatial branching [31]. In contrast, multitree methods typically solve mixed-integer linear programming (MILP) relaxations globally to obtain objective lower bounds, pushing the branching requirements to an efficient MILP solver [26]. The MILP relaxation can be refined with piecewise outer-approximation techniques (e.g., see [15]).

Both B&B and multitree algorithms require convex relaxations of the original MINLP. Convex relaxations are typically generated either with factorable programming (e.g., in BARON and ANTIGONE [29, 33]), using generalized McCormick envelopes (e.g., in MAINGO and EAGO [34, 38]), with the αBB approach (e.g., see [2]), or with edge-concave relaxations (e.g., see [19]). The factorable programming approach introduces auxiliary variables in order to decompose constraints into terms that can be systematically relaxed. The following example illustrates this approach.

Example 1.1. Consider the constraint $z = \ln(w_1 w_2)$. The factorable programming approach would split this into two constraints:

$$z = \ln(w_3) \tag{1}$$
$$w_3 = w_1 w_2 \tag{2}$$

These two constraints could then be relaxed independently.

Generalized McCormick envelopes avoid the introduction of auxiliary variables by propagating rules [28, 36] through expression trees. For some problems, such as global optimization over neural networks, this can lead to drastic reductions in problem size, and corresponding improvements in computational performance [34]. The αBB approach generates convex relaxations by adding quadratic terms to functions until it can be proven that the augmented function is a convex underestimator of the original function. For example, Adjiman et al. [1] provide a lower bound on the minimum eigenvalue of the Interval Hessian. If the minimum eigenvalue of the Interval Hessian is larger than 0, then the function is guaranteed to be convex. Finally, for some problem classes, custom relaxations have been developed to exploit knowledge of the system being optimized [22].

Regardless of the method used to generate a convex relaxation, the tightness of the resulting convex relaxation depends heavily on variable bounds. Furthermore, the performance of global optimization solvers depends heavily on the tightness of the convex relaxation used to generate lower bounds. Tighter relaxations can be generated through bounds tightening (BT). Also referred to as domain reduction, BT involves determining valid variable bounds which are tighter than current variable bounds. BT can strengthen convex relaxations for the lower bounding problems, reduce the number of nodes visited in B&B trees, and reduce the required number of binary variables needed in MILP relaxations for multitree methods. For example, turning off domain reduction techniques in BARON, which is an MINLP B&B solver [35], results in a 47% increase in computational time and an 802% increase in number of nodes explored for problems in MINLPLib [6, 31]. Examples 1.2 and 1.3 illustrate how tightening variable bounds can tighten convex relaxations of nonconvex constraints.

Example 1.2. Consider a convex relaxation of $z{=}w^2$:

$$z \geq 2\underline{w}w - \underline{w}^2 \tag{3}$$

$$z \geq 2\overline{w}w - \overline{w}^2 \tag{4}$$

$$z \leq \overline{w}^2 + \frac{\overline{w}^2 - \underline{w}^2}{\overline{w} - \underline{w}}(w- \tag{5}$$

where $\underline{w}$ and $\overline{w}$ are the lower and upper bounds of w. Figure 1 illustrates the feasible region of this relaxation for different bounds on w. The solid line represents the feasible region of the original constraint, and the shaded region is the feasible region of the convex relaxation. The figure on the right has tighter bounds on w, so the feasible region of the convex relaxation on the right is much smaller (i.e., the relaxation is tighter).

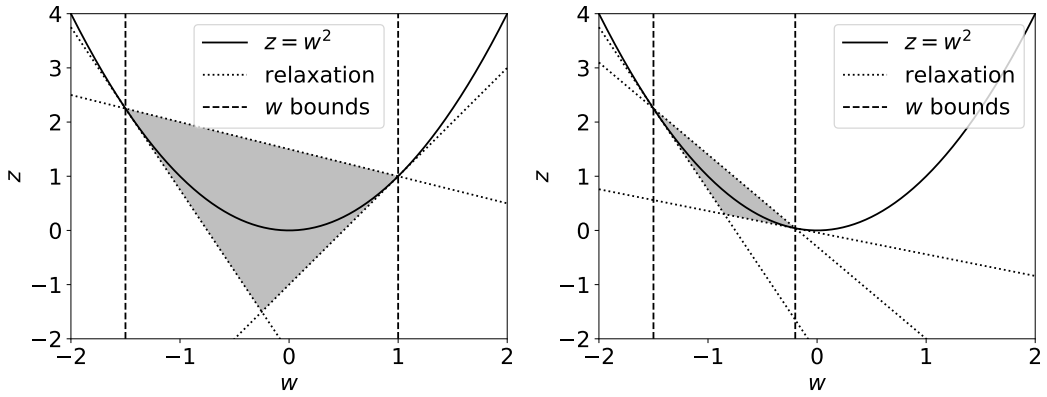

Figure 1: Linear relaxations of $z = w^2$ with different bounds on w.

Example 1.3. Consider the constraint $z \geq w_1 w_2$ with $\underline{w}_1 \leq w_1 \leq \overline{w}_1$ and $\underline{w}_2 \leq w_2 \leq \overline{w}_2$. The convex hull of this constraint is given by the McCormick envelopes for bilinear terms:

$$z \geq \underline{w}_1 w_2 + w_1 \underline{w}_2 - \underline{w}_1 \underline{w}_2 \tag{6}$$

$$z \geq \overline{w}_1 w_2 + w_1 \overline{w}_2 - \overline{w}_1 \overline{w}_2 \tag{7}$$

$$\tag{8}$$

The constraint violation, ϵ, obtained by using the relaxation is bounded by

$$\epsilon \leq \min\{w_1 w_2 - \underline{w}_1 w_2 - w_1 \underline{w}_2 + \underline{w}_1 \underline{w}_2, w_1 w_2 - \overline{w}_1 w_2 - w_1 \overline{w}_2 + \overline{w}_1 \overline{w}_2\} \tag{9}$$

This bound is maximized when $w_1 = \frac{1}{2}(\underline{w}_1 + \overline{w}_1)$ and $w_2 = \frac{1}{2}(\underline{w}_2 + \overline{w}_2)$, and the maximum constraint violation is given by [2]:

$$\epsilon_{\max} = \frac{1}{4}(\overline{w}_1 - \underline{w}_1)(\overline{w}_2 - \underline{w}_2) \tag{10}$$

Figure 2 shows how the maximum constraint violation, $\epsilon_{\max}$ changes with $(\overline{w}_1 - \underline{w}_1)$ and $(\overline{w}_2 - \underline{w}_2)$. Equation (10) and Figure 2 illustrate the effect of variable bounds on the tightness of the convex relaxation for $z \geq w_1 w_2$. The maximum constraint violation decreases quickly as the bound ranges, $(\overline{w}_1 - \underline{w}_1)$ and $(\overline{w}_2 - \underline{w}_2)$, decrease.

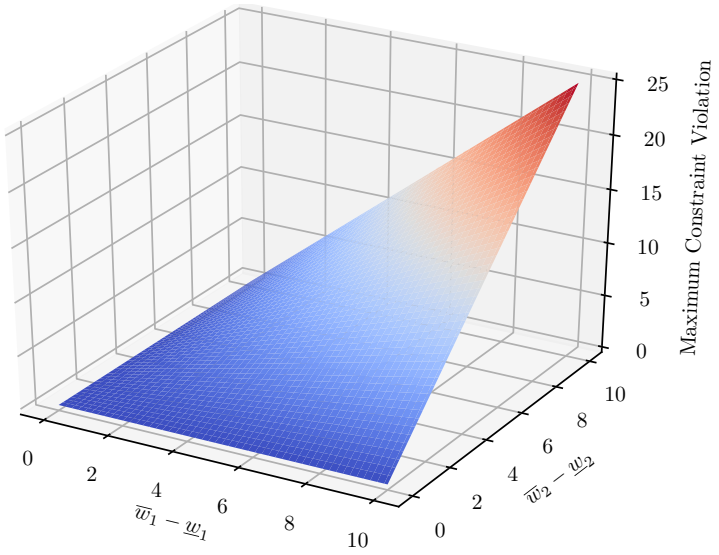

Figure 2: The effect of variable bounds on the maximum constraint violation that can be obtained when using McCormick envelopes to relax $z \geq w_1 w_2$.

Many BT techniques for MINLP's exist. Consider the following MINLP:

$$\min f(x) \tag{11a}$$
$$\text{s.t.}$$
$$h_c(x, y) \leq 0 \qquad \forall c \in \mathbb{C} \tag{11b}$$
$$\underline{x_v} \leq x_v \leq \overline{x_v} \qquad \forall v \in \mathcal{V}_x \tag{11c}$$
$$y_v \in \{0, 1\} \qquad \forall v \in \mathcal{V}_y \tag{11d}$$

where $\mathbb{C}$ is the set of constraints, $\mathcal{V}_x$ is the set of continuous variables, and $\mathcal{V}_y$ is the set of binary variables. For simplicity, we assume $f(x)$ is convex. Let the following be the convex relaxation of Problem (11):

$$\min f(x) \tag{12a}$$
$$\text{s.t.}$$
$$g_c(x, y) \leq 0 \qquad \forall c \in \mathcal{C} \tag{12b}$$
$$\underline{x_v} \leq x_v \leq \overline{x_v} \qquad \forall v \in \mathcal{V}_x \tag{12c}$$
$$0 \leq y_v \leq 1 \qquad \forall v \in \mathcal{V}_y \tag{12d}$$

where $\mathcal{C}$ is the set of constraints comprising convex underestimators of the original constraints, h. Here, each constraint, g_c, is convex. In other words, $\{(x, y) : g_c(x, y) \leq 0 \; \forall c \in \mathcal{C}\} \supseteq \{(x, y) : h_c(x, y) \leq 0 \; \forall c \in \mathbb{C}\}$ and is a convex set.

An effective but computationally expensive approach to bounds tightening is optimization-based bounds tightening (OBBT) [31]. OBBT typically involves solving two convex opti-

mization problems for each variable (or a subset of the variables):

$$\min / \max x_i \tag{13a}$$

s.t.

$$(12b), (12c), (12d) \tag{13b}$$

$$f(x) \leq U \tag{13c}$$

Here, U, is the best-known feasible solution to Problem (11). If the optimal objective of the minimization problem is larger than the current lower bound, $\underline{x_i}$, for the corresponding variable, then the lower bound can be updated. A similar argument holds for the upper bound and the maximization problem.

A very efficient approach to BT is feasibility-based bounds tightening (FBBT) and its extensions [4, 31]. FBBT can be performed by forming a directed acyclic graph (DAG) from the constraints of Problem (11) and using interval arithmetic to propagate bounds through the graph [30]. FBBT is usually less effective than OBBT because only one constraint is considered at a time [31]. Belotti et al. [4] present a linear program (LP) which converges to the fixed-point of the FBBT algorithm given the DAG is formed from a linear relaxation of Problem (11); as a result, a single LP solution enables bounds updates for all variables of Problem (11).

Ryoo and Sahinidis [32] utilize the Lagrange multipliers from the solution of Problem (12) to tighten bounds. Let L and U be the optimal objective value of Problem (12) and a valid upper bound for Problem (11), respectively. If a variable x_i is at its upper bound, $\overline{x_i}$, with an optimal multiplier value $\lambda^* > 0$ at the solution of Problem (12), then the lower bound may be updated as follows

$$\underline{x_i} = \overline{x_i} - \frac{U - L}{\lambda^*} \tag{14}$$

A similar argument holds if the variable is at its lower bound at the solution of Problem (12) [32]. This approach is also very efficient because it only requires the solution of Problem (12), which is typically solved at every node of the B&B tree. For a more thorough review of BT methods, see [31].

Although OBBT is computationally expesensive, it can be very effective for some problems. For example, OBBT has been shown to be very effective for refinement of optimal power flow (OPF) problems. Coffrin et al. [12] demonstrate that OBBT is very effective on polar relaxations of the optimal power flow problem. Relaxations of the rectangular form, tightened with OBBT, can also be quite effective if reference bus constraints are incorporated [8]. In fact, Bynum et al. [7] demonstrate that the effectiveness of OBBT at a given iteration is directly related to the topological distance of the corresponding variable from the reference bus. Multiple global optimization algorithms have been written for the OPF problem which rely heavily on OBBT [23, 25, 27].

Despite the effectiveness of OBBT for domain reduction, the approach is very computationally expensive, especially for large problems. A great deal of research has been done recently to mitigate the high computational cost, both for general MINLP's and for specific applications. Gleixner et al. [14] presented three improvements to OBBT. First, they presented an aggressive filtering approach which identifies variables for which it can be guaranteed that the bounds cannot be improved by solving an OBBT problem. This improves computational efficiency by reducing the number of OBBT problems solved while producing the same improvement in variable bounds. Second, the they compared different methods

for ordering the variables for which OBBT problems are solved. They found that a greedy ordering algorithm best utilizes simplex warmstarts, reducing the total number of simplex iterations spent in OBBT problems. Finally, they presented Lagrangian Variable Bounds, which are expressions generated from the dual information obtained from the solution of each OBBT problem. These expressions provided a mechanism to compute tightened variable bounds for one variable subsequent to tightening the bounds of other variables. They solved OBBT problems with the relaxation at the root node of the B&B tree. Then, when spatial branching occurred on one variable, the Lagrangian Variable Bounds could be used to efficiently update the bounds of other variables.

Custom approaches to BT have also been developed for specific applications. There has been a great deal of research on BT techniques for optimal power flow problems. For example, Chen et al. [11] present closed-form bounds tightening methods by writing the first order necessary conditions for minimizing or maximizing a variable subject to a small subset of constraints. Kocuk et al. [22] use an efficient form of OBBT for the OPF problem where, for each variable, a small optimization problem is solved by only considering a small portion of the electric grid and the corresponding constraints. In other words, a valid form of problem (13) is solved for each variable, but only a carefully selected subset of the constraints are included. There is a tradeoff between the strength of the bounds obtained and the computational effort required because, as the size of the network considered is reduced, information on feasibility of the remainder of the network is lost.

Despite recent advancements, OBBT generally remains computationally expensive. Figure 3 shows the time required to perform a single iteration of OBBT for problems from MINLPLib and the Power Grid Lib - Optimal Power Flow library as a function of problem size. Note that the figures were generated with an implementation that utilizes the aggressive filtering approach described in [14]. As the figures show, the computational effort

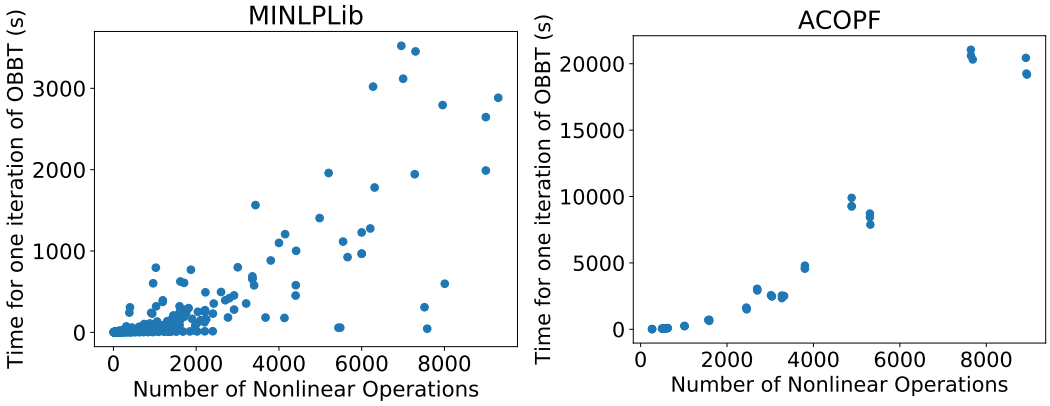

Figure 3: Wallclock time required for a single iteration of OBBT as a function of the number of nonlinear operations in the problem. The figure on the left shows results for a subset of test problems from MINLP-LIB (http://www.minlplib.org). The figure on the right shows results for a subset of test problems from the Power-Grid-Lib Optimal Power Flow repository (https://github.com/power-grid-lib/pglib-opf) [3].

can grow very quickly with problem size. In this paper, we propose a decomposed bounds tightening (DBT) algorithm based on graph partitioning, generalizing and extending the BT ideas for OPF presented by Kocuk et al. [22]. The main idea is to solve a few large

OBBT problems to tighten the bounds on a set of linking variables introduced during the graph partitioning stage and many small OBBT problems to tighten the nonlinear variables (whose bounds impact the tightness of the relaxation). As discussed in more detail below, by solving a few large bounds tightening problems, we retain feasibility information from the constraints not considered when solving the small bounds tightening problems. We demonstrate the effectiveness of the algorithm on OPF test cases, and test problems from MINLPLib.

The remainder of this paper is outlined as follows. In the following section we present our proposed DBT algorithm in detail. We then describe the test problems in detail. Finally, we present numerical results and summarize the paper.

2 Decomposed Bounds Tightening

In this section, we present our decomposed bounds tightening (DBT) algorithm. The basis of the algorithm is that small optimization problems can be solved efficiently. One way to solve small BT problems is to solve the OBBT problems (13) but with most of the constraints discarded. Although this does improve computational performance, it produces weaker bounds because the feasible region has been enlarged. The DBT algorithm attempts to retain information from the removed constraints by first solving OBBT problems with the full set of constraints but only for a small subset of the variables. This idea will be explained further after a few definitions and a more formal description of the algorithm.

For clarity, we rewrite Problem (11) as follows:

$$\min_x f(x) \tag{15a}$$

s.t.

$$a_i^T x \leq b_i \qquad \forall i \in \mathcal{C}^L \tag{15b}$$

$$p_i h_i(x) \leq q_i^T x \qquad \forall i \in \mathcal{C}^{NL} \tag{15c}$$

$$\underline{x_i} \leq x_i \leq \overline{x_i} \qquad \forall i \in \mathcal{V} \tag{15d}$$

$$x_i \in \{0,1\} \qquad \forall i \in \mathcal{V}^B \tag{15e}$$

where $f(x)$ is convex, $\mathcal{C}^L$ is the set of linear constraint indices, $\mathcal{C}^{NL}$ is the set of nonlinear constraint indices, $\mathcal{V}$ is the set of variable indices, and $\mathcal{V}^B$ is the set of binary variable indices (with $\mathcal{V}^B \subseteq \mathcal{V}$). Additionally, $a_i \in \mathbb{R}^{|\mathcal{V}|}$ $\forall i \in \mathcal{C}^L$, $b \in \mathbb{R}^{|\mathcal{C}^L|}$, $p \in \mathbb{R}^{|\mathcal{C}^{NL}|}$, and $q_i \in \mathbb{R}^{|\mathcal{V}|}$ $\forall i \in \mathcal{C}^{NL}$ are parameters, and $\underline{x} \in \mathbb{R}^{|\mathcal{V}|}$ and $\overline{x} \in \mathbb{R}^{|\mathcal{V}|}$ are the lower and upper bounds of the variables, respectively. The nonlinear functions, $h(x)$, are one of the following univariate or bivariate functions:

- $x_i x_j$
- $\ln(x_i)$
- $\exp(x_i)$
- x_i^c; c is a positive even integer

- x_i^c; $\underline{x_i} > 0$
- $\cos(x_i)$; $-\frac{\pi}{2} \leq \underline{x_i} \leq \overline{x_i} \leq \frac{\pi}{2}$
- $\sin(x_i)$; $-\frac{\pi}{2} \leq \underline{x_i} \leq \overline{x_i} \leq \frac{\pi}{2}$
- $\arctan(x_i)$

The algorithm described below is implemented using Coramin (https://github.com/coramin/coramin), which supports relaxations of the above univariate and bivariate functions. Note that many MINLP's can be transformed into the form of Problem (15) through factorable programming and preprocessing transformations [29].

Let $\mathcal{C}_0 \equiv \mathcal{C}^L \cup \mathcal{C}^{NL}$ be the full set of constraints, and let $\mathcal{V}_0 \equiv \mathcal{V}$ be the full set of variables. Let $G_0 = (\mathcal{V}_0, \mathcal{C}_0, \mathcal{E}_0)$ be a bipartite graph where $\mathcal{V}_0$ is the set of vertices in one part and $\mathcal{C}_0$ is the set of vertices in the second part. Each edge in the set of edges, $\mathcal{E}_0$, connects verex $v \in \mathcal{V}_0$ to vertex $c \in \mathcal{C}_0$ if and only if variable x_v appears in the constraint indexed by c with a nonzero coefficient or is a term in $h_c(x)$.

The algorithm begins by partitioning G_0 into two sub-graphs, $\widehat{G_1} = (\widehat{\mathcal{V}_1}, \widehat{\mathcal{C}_1}, \widehat{\mathcal{E}_1})$ and $\widehat{G_2} = (\widehat{\mathcal{V}_2}, \widehat{\mathcal{C}_2}, \widehat{\mathcal{E}_2})$, using a balanced partitioning algorithm. A balanced partitioning algorithm seeks to minimize the number of edges that need to be removed in order to partition a graph into two graphs with a roughly equal number of vertices. In other words, $\mathcal{V}_0 = \widehat{\mathcal{V}_1} \cup \widehat{\mathcal{V}_2}$, $\mathcal{C}_0 = \widehat{\mathcal{C}_1} \cup \widehat{\mathcal{C}_2}$, $\widehat{\mathcal{E}_1} \cup \widehat{\mathcal{E}_2} \subseteq \mathcal{E}_0$, $\widehat{\mathcal{V}_1} \cap \widehat{\mathcal{V}_2} = \emptyset$, $\widehat{\mathcal{C}_1} \cap \widehat{\mathcal{C}_2} = \emptyset$, $\widehat{\mathcal{E}_1} \cap \widehat{\mathcal{E}_2} = \emptyset$, $\widehat{\mathcal{V}_1} \cup \widehat{\mathcal{C}_1} \neq \emptyset$, and $\widehat{\mathcal{V}_2} \cup \widehat{\mathcal{C}_2} \neq \emptyset$. This procedure is illustrated in the first two columns of Figure 4.

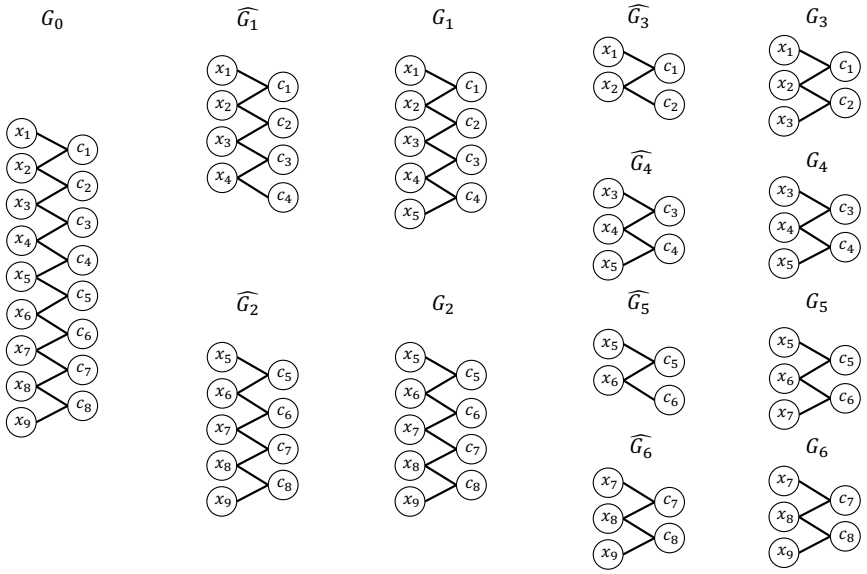

Figure 4: Partitioning procedure

Algorithm 1 Graph Partitioning

function PARTITION($G = (\mathcal{V}, \mathcal{C}, \mathcal{E})$)
  Partition G into two graphs, $\widehat{G_a} = (\widehat{\mathcal{V}_a}, \widehat{\mathcal{C}_a}, \widehat{\mathcal{E}_a})$ and $\widehat{G_b} = (\widehat{\mathcal{V}_b}, \widehat{\mathcal{C}_b}, \widehat{\mathcal{E}_b})$ such that $\widehat{\mathcal{V}_1} \cup \widehat{\mathcal{C}_1} \neq \emptyset$ and $\widehat{\mathcal{V}_2} \cup \widehat{\mathcal{C}_2} \neq \emptyset$
  $\mathcal{E}^R = \mathcal{E} - (\widehat{\mathcal{E}_a} \cup \widehat{\mathcal{E}_b})$
  $\mathcal{V}^R = \{v : (v, c) \in \mathcal{E}^R, v \in \mathcal{V}\}$
  $\mathcal{V}_a = \widehat{\mathcal{V}_a} \cup (\mathcal{V}^R \cap \widehat{\mathcal{V}_b})$
  $\mathcal{V}_b = \widehat{\mathcal{V}_b} \cup (\mathcal{V}^R \cap \widehat{\mathcal{V}_a})$
  $\mathcal{C}_a = \widehat{\mathcal{C}_a}$
  $\mathcal{E}_a = \widehat{\mathcal{E}_a} \cup \{(v, c) : v \in (\mathcal{V}^R \cap \widehat{\mathcal{V}_b}),\ (v, c) \in \mathcal{E}^R\}$
  $\mathcal{C}_b = \widehat{\mathcal{C}_b}$
  $\mathcal{E}_b = \widehat{\mathcal{E}_b} \cup \{(v, c) : v \in (\mathcal{V}^R \cap \widehat{\mathcal{V}_a}),\ (v, c) \in \mathcal{E}^R\}$
  $G_a = (\mathcal{V}_a, \mathcal{C}_a, \mathcal{E}_a)$
  $G_b = (\mathcal{V}_b, \mathcal{C}_b, \mathcal{E}_b)$
end function

The graphs $\widehat{G_1}$ and $\widehat{G_2}$ are not well defined in the sense that some constraints in the graphs utilize variables that are not in the same graph. In the example in Figure 4, $\widehat{G_1}$ contains constraint c_4 which depends on variable x_5 even though x_5 is not in $\widehat{G_1}$. Therefore, we define new graphs, G_1 and G_2, with some vertices duplicated in both graphs. Let $\mathcal{E}_0^R = \mathcal{E}_0 - (\widehat{\mathcal{E}_1} \cup \widehat{\mathcal{E}_2})$ be the set of edges removed in order to partition the graph G_0. Let $\mathcal{V}_0^R = \{v : (v, c) \in \mathcal{E}_0^R, v \in \mathcal{V}_0\}$ be the set of variable indices in $\mathcal{V}_0$ associated with removed edges. Let $\mathcal{V}_1 = \widehat{\mathcal{V}_1} \cup (\mathcal{V}_0^R \cap \widehat{\mathcal{V}_2})$. Similarly, let $\mathcal{V}_2 = \widehat{\mathcal{V}_2} \cup (\mathcal{V}_0^R \cap \widehat{\mathcal{V}_1})$. Additionally, $\mathcal{C}_1 = \widehat{\mathcal{C}_1}$, $\mathcal{E}_1 = \widehat{\mathcal{E}_1} \cup \{(v, c) : v \in (\mathcal{V}_0^R \cap \widehat{\mathcal{V}_2}),\ (v, c) \in \mathcal{E}_0^R\}$, $\mathcal{C}_2 = \widehat{\mathcal{C}_2}$ and $\mathcal{E}_2 = \widehat{\mathcal{E}_2} \cup \{(v, c) : v \in (\mathcal{V}_0^R \cap \widehat{\mathcal{V}_1}),\ (v, c) \in \mathcal{E}_0^R\}$. Finally, $G_1 = (\mathcal{V}_1, \mathcal{C}_1, \mathcal{E}_1)$ and $G_2 = (\mathcal{V}_2, \mathcal{C}_2, \mathcal{E}_2)$. This procedure is illustrated in the third column of Figure 4 and formalized in Algorithm 1.

We define the feasible region associated with bipartite graph G_j, $\mathcal{F}(G_j)$, as

$$\begin{aligned}\mathcal{F}(G_j) \equiv \{x : & a_i^T x \leq b_i \ \ \forall i \in \mathcal{C}_j^L, \\ & p_i h_i(x) \leq q_i^T x \ \ \forall i \in \mathcal{C}_j^{NL}, \\ & \underline{x_i} \leq x_i \leq \overline{x_i} \ \ \forall i \in \mathcal{V}_j, \\ & x_i \in \{0, 1\} \ \ \forall i \in \mathcal{V}_j^B\}. \end{aligned} \tag{16}$$

Note that $\mathcal{F}(G_0) \subseteq \mathcal{F}(G_1)$ and $\mathcal{F}(G_0) \subseteq \mathcal{F}(G_2)$. Additionally, $\mathcal{F}(G_0) = \mathcal{F}(G_1) \cap \mathcal{F}(G_2)$. We also define the following convex relaxation.

$$\begin{aligned}\underline{\mathcal{F}}(G_j) \equiv \{x : & a_i^T x \leq b_i \ \ \forall i \in \mathcal{C}_j^L, \\ & p_i \underline{h_i}(x) \leq q_i^T x \ \ \forall i \in \mathcal{C}_j^{NL}, \\ & \underline{x_i} \leq x_i \leq \overline{x_i} \ \ \forall i \in \mathcal{V}_j\}. \end{aligned} \tag{17}$$

where $\underline{h_i}(x)$ is a convex relaxation of $h_i(x)$. Note that $\mathcal{F}(G_0) \subseteq \mathcal{F}(G_1) \subseteq \underline{\mathcal{F}}(G_1)$ and $\mathcal{F}(G_0) \subseteq \mathcal{F}(G_2) \subseteq \underline{\mathcal{F}}(G_2)$.

The DBT algorithm performs OBBT on the shared variables in $\mathcal{V}_0^R$ with the full set of

constraints, $\mathcal{C}_0$. In other words, problems of the form

$$\min / \max x_i \tag{18a}$$
$$\text{s.t.}$$
$$x \in \underline{\mathcal{F}}(G_0) \tag{18b}$$
$$f(x) \leq U \tag{18c}$$

are solved for all $i \in \mathcal{V}_0^R$, allowing the variable bounds, $\underline{x}$ and $\overline{x}$, to be updated. OBBT problems are then solved with the smaller sets of constraints, $\mathcal{C}_1$ and $\mathcal{C}_2$, and the updated variable bounds. Let $\mathcal{V}_j^{NL}$ be the nonlinear variables, or the set of variables which appear in the nonlinear functions, h_i, such that $i \in \mathcal{C}_j^{NL}$. Additionally, let $\mathcal{V}_j^{NC} = \mathcal{V}_j^{NL} \cup \mathcal{V}_j^B$. Then, OBBT problems of the form

$$\min / \max x_i \tag{19a}$$
$$\text{s.t.}$$
$$x \in \underline{\mathcal{F}}(G_1) \tag{19b}$$

are solved for all $i \in \mathcal{V}_1^{NC}$ and problems of the form

$$\min / \max x_i \tag{20a}$$
$$\text{s.t.}$$
$$x \in \underline{\mathcal{F}}(G_2) \tag{20b}$$

are solved for all $i \in \mathcal{V}_2^{NC}$. The optimality cut (Equation (18c)) is not included in Problem (19) or in Problem (20) because it is possible that some of the variables in the objective do not appear in any of the constraints in each of these problems. Depending on the variable bounds, including the optimality cut could still be beneficial, but it is more likely to increase the problem size without any benefit.

The DBT algorithm should be significantly faster than OBBT if $|\mathcal{V}_0^R| \ll |\mathcal{V}_0^{NC}|$ and if the sizes of Problems (19) and (20) are significantly smaller than the size of Problem (18), which occurs if the partitions G_1 and G_2 are balanced (i.e., $|\mathcal{V}_1 \cup \mathcal{C}_1| \simeq |\mathcal{V}_2 \cup \mathcal{C}_2|$ and $|\mathcal{E}_1| \simeq |\mathcal{E}_2|$). However, depending on $|\mathcal{V}_1^{NC}|$ and $|\mathcal{V}_2^{NC}|$, Problems (19) and (20) may still be prohibitively expensive. If that is the case, graphs G_1 and G_2 can be further partitioned. In fact, the procedure described above can be performed recursively, producing a binary tree of graphs and their associated OBBT problems as illustrated in the last two columns of Figure 4.

The binary tree of graphs produced by recursive partitioning is illustrated in Figure 5. At leaf nodes of this tree (nodes without any children), nonlinear and binary variables are tightened. At non-leaf nodes of the tree, only coupling variables introduced by the partitioning procedure are tightened. Additionally, at each node except the root node (G_0), the bounds on the coupling variables obtained from parent nodes are utilized, which provide feasibility information from the constraints left out when performing bounds tightening at that node.

If partitioning is performed recursively, a stopping criteria must be defined. We utilize three stopping criteria. First, we do not partition a graph, $G = (\mathcal{V}, \mathcal{C}, \mathcal{E})$, if the the number of nonzeros in the Jacobian of the constraints, $\mathcal{E}$, is below a threshold, $\underline{E}$. Second, we limit the number of stages of partitioning to $\overline{S}$, or equivalently the depth of the partitioning tree. Finally, it is possible that so many edges need removed in order to partition the graph that

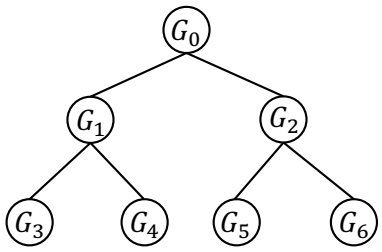

Figure 5: Tree of problem graphs produced by recursive partitioning

$|\mathcal{V}^R|$ is large and decomposition is not beneficial. Suppose graph $G = (\mathcal{V}, \mathcal{C}, \mathcal{E})$ is partitioned into graphs $G_a = (\mathcal{V}_a, \mathcal{C}_a, \mathcal{E}_a)$ and $G_b = (\mathcal{V}_b, \mathcal{C}_b, \mathcal{E}_b)$. We define the following Partitioning Ratio, P^R.

$$P^R \equiv \frac{|\mathcal{V}^{NC}||\mathcal{E}|}{|\mathcal{V}^R||\mathcal{E}| + |\mathcal{V}_a^{NC}||\mathcal{E}_a| + |\mathcal{V}_b^{NC}||\mathcal{E}_b|} \tag{21}$$

The numerator is a measure of the computational effort required to perform OBBT if graph G is not partitioned. The first term, $|\mathcal{V}^{NC}|$, is the number of variables for which OBBT problems need to be solved. The second term, $|\mathcal{E}|$, is the number of nonzeros in the Jacobian of the constraints and is a measure of the problem size. Similarly, the denominator is a measure of the the computational effort required if partitioning is performed. The larger the value of P^R, the more valuable partitioning is. On the other hand, if $P^R \leq 1$, then no benefit would be expected from decomposition.

It is worth noting that, if a bipartite graph, $G = (\mathcal{V}, \mathcal{C}, \mathcal{E})$ is partitioned into graphs $G_a = (\mathcal{V}_a, \mathcal{C}_a, \mathcal{E}_a)$ and $G_b = (\mathcal{V}_b, \mathcal{C}_b, \mathcal{E}_b)$ using Algorithm 1, and if the resulting graphs are roughly balanced (i.e., $|\mathcal{V}_a^{NC}| \approx |\mathcal{V}_b^{NC}|$ or $|\mathcal{E}_a| \approx |\mathcal{E}_b|$), then the maximum Partition Ratio, P^R, defined by Equation (21) is 2. It is possible to obtain partitioning ratios higher than 2 if the majority of the edges are placed in one graph and the majority of the nonconvex variables are placed in the other graph. For example, if a partition can be obtained such that $\frac{|\mathcal{V}_a^{NC}|}{|\mathcal{V}^{NC}|} \approx 0$, $\frac{|\mathcal{E}_a|}{|\mathcal{E}|} \approx 1$, $\frac{|\mathcal{V}_b^{NC}|}{|\mathcal{V}^{NC}|} \approx 1$, $\frac{|\mathcal{E}_b|}{|\mathcal{E}|} \approx 0$, then the partitioning ratio can be quite large. Our implementation utilizes a balanced partitioning ratio, so we typically obtain partitioning ratios less than 2. Partitioning schemes that achieve higher partitioning ratios may be beneficial, but the impact on the resulting variable bounds is unclear and should be explored (including methods with n-ary partitions rather than binary partitions).

We can now formalize the Decomposed Bounds Tightening (DBT) algorithm in Algorithm 2. Here, $\overline{S}$ is the maximum number of stages, $\underline{E}$ is the minimum number of edges to perform partitioning, $\underline{P^R}$ is the minimum Partition Ratio required to accept a partition, N is the maximum number of bounds tightening iterations, s is the stage of the binary tree, $\mathcal{T}_s$ is the set of graphs in stage s, $\mathcal{L}$ is the set of leaves (or graphs that are not partitioned), and n is the bounds tightening iteration. Lines 3 – 19 perform graph partitioning, and lines 22 – 48 perform bounds tightening. In the example in Figure 5, graph G_0 is in stage 0, graphs G_1 and G_2 are in stage 1, and graphs G_3, G_4, G_5, and G_6 are in stage 2.

The while condition on line 3 checks to see if there are any graphs in stage s. If so, then the problem size of each graph in stage s is checked on line 6. If the graph is large and we have not exceeded the maximum number of stages, then the graph is partitioned using Algorithm 1. The Partitioning Ratio is then computed on line 8, and the partition

is accepted if the Partitioning Ratio is larger than the minimum Partitioning Ratio. If the Partitioning Ratio is not large enough, the graph is not large enough, or the maximum number of stages has been exceeded, then the graph is added to the set of leaf graphs.

Bounds tightening is performed on lines 22 – 48 for N iterations. For each stage and each graph in each stage, we check if the graph is in the set of leaf graphs on line 26. If so, bounds tightening is performed on all of the nonlinear and binary variables associated with the graph and only subject to the subset of constraints associated with the graph. If the graph is not in the set of leaf graphs, then bounds tightening is performed on all of the variables associated with edges that were removed in order to partition the graph. These are the "coupling" variables. By tightening these "coupling" variables (hopefully few in number), we retain some feasibility information from the constraints left out when performing bounds tightening with subsequent child graphs.

We can also define an Overall Partitioning Ratio, P^{OR}, as

$$P^{OR} \equiv \frac{|\mathcal{V}_0^{NC}||\mathcal{E}_0|}{\sum_{s=0}^{S}\sum_{i\in\mathcal{T}_s, i\notin\mathcal{L}}(|\mathcal{V}_i^{R}||\mathcal{E}_i|) + \sum_{i\in\mathcal{L}}(|\mathcal{V}_i^{NC}||\mathcal{E}_i|)} \tag{22}$$

which is a measure of the overall effectiveness of a given recursive partitioning.

2.1 Tightening Leaves Only

An obvious alternative algorithm to the DBT algorithm can be developed by only tightening the variables in the set $\mathcal{V}^{NC}$ and excluding the coupling variables. The result is that bounds tightening problems are only solved with the small problems defined by the leaf graphs. We denote this algorithm the Leaves algorithm and formalize it in Algorithm 3. Note that the Leaves algorithm is a generalization of the bounds tightening procedure developed by Kocuk et al. [22] for optimal power flow problems.

The motivation for still requiring a minimum Partitioning Ratio, $\underline{P^R}$, is that a large number of removed edges in the graph partitioning is an indication of a high degree of coupling which implies that the relaxations used in the bounds tightening problems will be very weak, producing poor bounds.

Algorithm 2 DBT

function DBT(G_0, $\overline{S}$, $\underline{E}$, $\underline{P^R}$, N, U)
  $s \leftarrow 0$, $\mathcal{T}_0 \leftarrow \{G_0\}$, $\mathcal{L} \leftarrow \varnothing$
  while $\mathcal{T}_s \neq \varnothing$ **do**
    $\mathcal{T}_{s+1} = \varnothing$
    for $G_i = (\mathcal{V}_i, \mathcal{C}_i, \mathcal{E}_i) \in \mathcal{T}_s$ **do**
      if $|\mathcal{E}_i| \geq \underline{E}$ and $s < \overline{S}$ **then**
        Partition G_i into graphs G_a and G_b using Algorithm 1
        Compute P^R from Equation (21)
        if $P^R \geq \underline{P^R}$ **then**
          $\mathcal{T}_{s+1} = \mathcal{T}_{s+1} \cup \{G_a, G_b\}$
        else
          $\mathcal{L} = \mathcal{L} \cup \{G_i\}$
        end if
      else
        $\mathcal{L} = \mathcal{L} \cup \{G_i\}$
      end if
    end for
    $s = s + 1$
  end while
  $S = s$
  $n = 0$
  while $n < N$ **do**
    $s = 0$
    while $s < S$ **do**
      for $G_i = (\mathcal{V}_i, \mathcal{C}_i, \mathcal{E}_i) \in \mathcal{T}_s$ **do**
        if $G_i \in \mathcal{L}$ **then**
          if $G_i = G_0$ **then**
            $\underline{x}_j = \min x_j$ s.t. $x \in \{x : x \in \underline{\mathcal{F}}(G_i), f(x) \leq U\}$ $\forall j \in \mathcal{V}_i^{NC}$
            $\overline{x}_j = \max x_j$ s.t. $x \in \{x : x \in \underline{\mathcal{F}}(G_i), f(x) \leq U\}$ $\forall j \in \mathcal{V}_i^{NC}$
          else
            $\underline{x}_j = \min x_j$ s.t. $x \in \underline{\mathcal{F}}(G_i)$ $\forall j \in \mathcal{V}_i^{NC}$
            $\overline{x}_j = \max x_j$ s.t. $x \in \underline{\mathcal{F}}(G_i)$ $\forall j \in \mathcal{V}_i^{NC}$
          end if
        else
          if $G_i = G_0$ **then**
            $\underline{x}_j = \min x_j$ s.t. $x \in \{x : x \in \underline{\mathcal{F}}(G_i), f(x) \leq U\}$ $\forall j \in \mathcal{V}_i^{R}$
            $\overline{x}_j = \max x_j$ s.t. $x \in \{x : x \in \underline{\mathcal{F}}(G_i), f(x) \leq U\}$ $\forall j \in \mathcal{V}_i^{R}$
          else
            $\underline{x}_j = \min x_j$ s.t. $x \in \underline{\mathcal{F}}(G_i)$ $\forall j \in \mathcal{V}_i^{R}$
            $\overline{x}_j = \max x_j$ s.t. $x \in \underline{\mathcal{F}}(G_i)$ $\forall j \in \mathcal{V}_i^{R}$
          end if
        end if
      end for
      $s = s + 1$
    end while
    Update the convex relaxations, $\underline{h}(x)$, based on new variable bounds
    $n = n + 1$
  end while
end function

Algorithm 3 Leaves

function DBT(G_0, $\overline{S}$, $\underline{E}$, $\underline{P^R}$, N, U)
  $s \leftarrow 0$, $\mathcal{T}_0 \leftarrow \{G_0\}$, $\mathcal{L} \leftarrow \varnothing$
  while $\mathcal{T}_s \neq \varnothing$ **do**
    $\mathcal{T}_{s+1} = \varnothing$
    for $G_i = (\mathcal{V}_i, \mathcal{C}_i, \mathcal{E}_i) \in \mathcal{T}_s$ **do**
      if $|\mathcal{E}_i| \geq \underline{E}$ and $s < \overline{S}$ **then**
        Partition G_i into graphs G_a and G_b using Algorithm 1
        Compute P^R from Equation (21)
        if $P^R \geq \underline{P^R}$ **then**
          $\mathcal{T}_{s+1} = \mathcal{T}_{s+1} \cup \{G_a, G_b\}$
        else
          $\mathcal{L} = \mathcal{L} \cup \{G_i\}$
        end if
      else
        $\mathcal{L} = \mathcal{L} \cup \{G_i\}$
      end if
    end for
    $s = s + 1$
  end while
  $S = s$
  $n = 0$
  while $n < N$ **do**
    $s = 0$
    while $s < S$ **do**
      for $G_i = (\mathcal{V}_i, \mathcal{C}_i, \mathcal{E}_i) \in \mathcal{T}_s$ **do**
        if $G_i \in \mathcal{L}$ **then**
          if $G_i = G_0$ **then**
            $\underline{x}_j = \min x_j$ s.t. $x \in \{x : x \in \underline{\mathcal{F}}(G_i), f(x) \leq U\}$ $\forall j \in \mathcal{V}_i^{NC}$
            $\overline{x}_j = \max x_j$ s.t. $x \in \{x : x \in \underline{\mathcal{F}}(G_i), f(x) \leq U\}$ $\forall j \in \mathcal{V}_i^{NC}$
          else
            $\underline{x}_j = \min x_j$ s.t. $x \in \underline{\mathcal{F}}(G_i)$ $\forall j \in \mathcal{V}_i^{NC}$
            $\overline{x}_j = \max x_j$ s.t. $x \in \underline{\mathcal{F}}(G_i)$ $\forall j \in \mathcal{V}_i^{NC}$
          end if
        end if
      end for
      $s = s + 1$
    end while
    Update the convex relaxations, $\underline{h}(x)$, based on new variable bounds
    $n = n + 1$
  end while
end function

Algorithm 4 FS (Full-Space OBBT)

function FS(G_0, N, U)
  $n = 0$
  while $n < N$ **do**
    $\underline{x}_j = \min x_j$ s.t. $x \in \{x : x \in \underline{\mathcal{F}}(G_0), f(x) \leq U\}\ \forall j \in \mathcal{V}_0^{NC}$
    $\overline{x}_j = \max x_j$ s.t. $x \in \{x : x \in \underline{\mathcal{F}}(G_0), f(x) \leq U\}\ \forall j \in \mathcal{V}_0^{NC}$
    Update the convex relaxations, $\underline{h}(x)$, based on new variable bounds
    $n = n + 1$
  end while
end function

3 Implementation Details

In the following section, we compare both the DBT and Leaves algorithms (Algorithm 2 and Algorithm 3, respectively) to the standard, full-space (FS) algorithm. For completeness, the FS algorithm is presented in Algorithm 4. All three algorithms have been implemented in Coramin (https://github.com/coramin/coramin). Coramin is an open-source Python package built on Pyomo [17, 18] that contains tools for developing global optimization algorithms. Coramin contains relaxations for constraints containing bilinear, univariate convex, univariate concave, cosine, sine, and arctangent terms along with classes for managing and refining these relaxations. These classes inherit from Pyomo Blocks for seamless integration with Pyomo models. Coramin also contains functions for generating convex relaxations of general Pyomo models using factorable programming. Additionally, Coramin contains domain reduction tools, including a parallel implementation of OBBT and the aggressive filtering technique for OBBT developed by Gleixner et al. [14]. Coramin makes direct use of Pyomo's persistent solver interfaces to efficiently handle model updates such as changing the objective for OBBT and updating convex relaxations.

We implemented the DBT algorithm described in the previous section within Coramin along with examples of its use. Custom Pyomo Blocks were developed in order to manage the hierarchy imposed by graph partitioning. Because Pyomo Blocks can be solved independently, the model can be constructed once according to the structure imposed during the first part of Algorithm 2. The leaf Blocks (the models defined by the leaf graphs) can be combined on parent Blocks to describe the larger models.

Our implementation begins by creating a Pyomo model for the problem of interest. For the ACOPF test cases, we utilize Egret (https://github.com/grid-parity-exchange/egret) to parse the Matpower files from the Power Grid Lib - Optimal Power Flow (PGLib-OPF) repository (https://github.com/power-grid-lib/pglib-opf) [3]. For the problems from MINLPLib (http://www.minlplib.org), we use Suspect [9] to parse OSIL files and create the corresponding Pyomo models. Next, FBBT is used with the nonlinear model to tighten variable bounds. Then, we check to see if the objective uses a single variable, $ObjVar$. If the variable $ObjVar$ is only used in a single constraint,

$$f(x) = ObjVar, \tag{23}$$

then we remove $ObjVar$ from the problem and replace the objective with $f(x)$. The reason for this is that, for many problems, including the objective as a constraint as shown in Equation (23) causes the bipartite graph to be far more dense, causing our partitioning method

to result in partitions with low Partitioning Ratios. Next, the problem is converted to the form of Problem (15) using the factorable programming implementation from Coramin.

At this point, a bipartite graph is formed from the problem's constraints, and Metis [21] is used to perform graph partitioning as described in the first part of Algorithm 2. We use a minimum Partition Ratio, $\underline{P^R}$, of 1.5. We set the maximum number of stages, $\overline{S}$, to

$$\overline{S} = \log_{10}(|\mathcal{E}_0|) \tag{24}$$

and the minimum number of edges, $\underline{E}$, to 1,000. Note that Metis is extremely fast, and only a negligible fraction of time is spent performing graph partitioning. Our implementation calls Metis 10 times for each graph with different seeds and selects the best partition. Finally, each nonconvex function, $h_i(x)$ is relaxed using Coramin.

We solve all bounds tightening problems with Gurobi [16]. Because some of the problems from MINLPLib were poorly conditioned, we disabled Gurobi's presolve. Note that this is not required for most problems. We set the Thread count to 1 and used dual-simplex for problems from MINLPLib and the barrier method for the problems from the PGLib-OPF repository. We set the optimality tolerance to 10^{-9} for the dual-simplex method and the barrier convergence tolerance to 10^{-9} for the barrier method. We also disabled crossover for the barrier method. These settings reduced round-off error in the variable bounds introduced by OBBT. The barrier method was utilized for the PGLib-OPF repository because it was reliable for those problems and significantly faster than the simplex methods. The dual-simplex method was utilized for the problems from MINLPLib because the barrier method occasionally ran into numerical trouble. A more sophisticated implementation could try both barrier and simplex for the first iteration and use the better method in subsequent passes.

Additional measures were taken to avoid further ill-conditioning as the BT algorithms progressed. First, if a variable's bounds are initially infinite, tightened bounds are only accepted if the abolute value of the tightened bound is less than 10^7. Otherwise, the bounds are unchanged (infinite). Second, variables are not tightened if the variable's upper and lower bounds are within 10^{-4} of each other. Furthermore, tightened bounds are relaxed slightly to account for solver tolerances. Let $\underline{x_i}$ be the current lower bound for x_i, and let $\underline{\widehat{x_i}}$ be the lower bound computed from one of the OBBT problems such as Problem (18). We set the new lower bound according to $\underline{x_i} = \max(\underline{x_i}, \underline{\widehat{x_i}} - 10^{-3})$. Similarly, $\overline{x_i} = \min(\overline{x_i}, \overline{\widehat{x_i}} + 10^{-3})$. Additionally, constraints generated from relaxing nonlinear constraints were removed from the problem if they introduced large matrix coefficients into the problem. This occurs, for example, when relaxing constraints involving exponential terms. Consider a relaxation of $y \geq \exp(x)$ with $-100 \leq x \leq 100$. Outer Approximation cuts at the bounds of x result in very poorly conditioned constraints. Excluding these poorly conditioned constraints only further relaxes the problem.

For the bounds tightening portion of all three algorithms, we utilize the aggressive filtering algorithm developed by Gleixner et al. [14]. We also utilize Pyomo's persistent solver interface to Gurobi for efficiently changing the objective and resolving.

Before each iteration of bounds tightening, we solve the following MILP relaxation to

obtain a candidate integer solution.

$$\min_x f(x) \tag{25a}$$

s.t.

$$a_i^T x \leq b_i \qquad \forall i \in \mathcal{C}^L \tag{25b}$$

$$p_i \underline{h_i}(x) \leq q_i^T x \qquad \forall i \in \mathcal{C}^{NL} \tag{25c}$$

$$\underline{x_i} \leq x_i \leq \overline{x_i} \qquad \forall i \in \mathcal{V} \tag{25d}$$

$$x_i \in \{0, 1\} \qquad \forall i \in \mathcal{V}^B \tag{25e}$$

The binary variables are then fixed to this candidate integer solution, and a continuous NLP is solved with IPOPT [37] to obtain an upper bound, UB, for use in the bounds tightening algorithms. We place a 10-minute time limit on the solution of each lower bounding problem and each upper bounding problem. We limit the bounds tightening algorithms to 10 iterations and 1 hour, excluding time for the lower and upper bounding problems. For some of the larger test problems, even a single iteration of the FS approach does not complete within an hour. In order to get data for at least one iteration, we extend the time limit for the first iteration to 7 hours.

4 Computational Results

In this section, we present computational results comparing the full-space (FS), Leaves, and decomposed (DBT) bounds tightening algorithms (Algorithms 4, 3, and 2, respectively). We tested each algorithm on both alternating current optimal power flow (ACOPF) problems from the PGLib-OPF repository and on problems from MINLPLib. The ACOPF problem is a nonlinear programming problem (NLP) seeking to minimize the operating cost of a transmission system while satisfying customer power demand. The problem is defined by a set of buses connected by a set of transmission lines. Each bus may have a generator and/or load. The constraints describing power flow on the transmission lines are nonlinear and nonconvex. We utilize the Quadratic Convex (QC) relaxation of the polar form of the ACOPF problem strengthened with a second-order cone (SOC) constraint [20].

We limited our analysis to test problems with less than 10,000 nonlinear operations and less than 100,000 nonzeros in the Jacobian of the constraints after converting the problem to the form of Problem (15). We also excluded problems with nonlinear operations not supported by Suspect or Coramin, including `erf`, `gammaFn`, `signpower`, `min`, and `abs`. Finally, we excluded problems without objectives, problems with special ordered sets, and problems with semicontinuous variables. After excluding problems with these restrictions, 1,467 test problems remained from MINLPLib and 57 test problems remained from the PGLib-OPF repository. Of these, 684 problems from MINLPLib had less than 1,000 nonzeros in the Jacobian of the constraints, which is one of the stopping criteria for the partitioning procedure outlined in Section 3. Essentially, these problems were too small for decomposition, so they were excluded from the analysis. Additionally, 415 of the problems from MINLPLib resulted in Partitioning Ratios (PR's) less than 1.5. As a result, these problems were not partitioned. Of the 57 ACOPF test problems, only 6 had less than 1,000 nonzeros in the Jacobian. None resulted in PR's less than 1.5. Numerical errors were encountered on one test problem from MINLPLib. In the end, we tested the proposed algorithms on 367 problems from MINLPLib and 51 problems from the PGLib-OPF repository. Recall that we use

Metis to perform graph partitioning. Metis is extremely fast, so attempting decomposition and computing the PR is cheap relative to a global optimization algorithm. Furthermore, the PR can be computed before any bounds tightening is performed. Therefore, only a small amount of time is sacrificed if the PR is low and decomposition should not be performed. On the other hand, if the PR is high, enormous performance gains can be achieved through decomposition, as our results below demonstrate.

The computational results are summarized Figures 6, 7, 8, 9, and 10. Figures 6 and 7 show the time required to complete one iteration of BT for each of the test cases. The

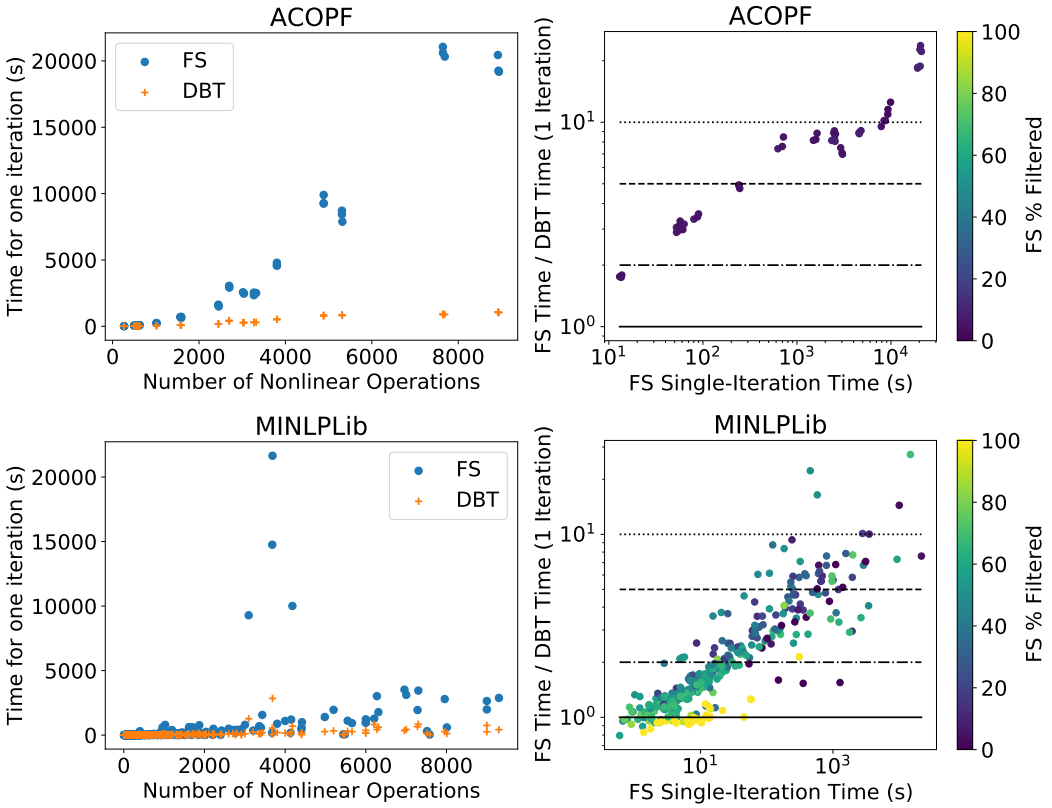

Figure 6: Comparison of time required for the FS and DBT algorithms. Note that these figures do not account for the effectiveness at tightening variable bounds - only the computational performance. The top two figures show results for the ACOPF test problems and the bottom two figures show results for the test problems from MINLPLib. The figures on the left show the time required for a single iteration of each algorithm as a function of the number of nonlinear operations in the problem. The figures on the right show the time required for an iteration of the FS algorithm divided by the time required for an iteration of the DBT algorithm. The figures on the right are also colored according the the percent of variables filtered (filtering was used in all algorithms). These figures highlight that the decomposition approach becomes increasingly important as the problem size and difficulty increase.

figures on the top of Figure 6 show results for the ACOPF test problems. The figures on

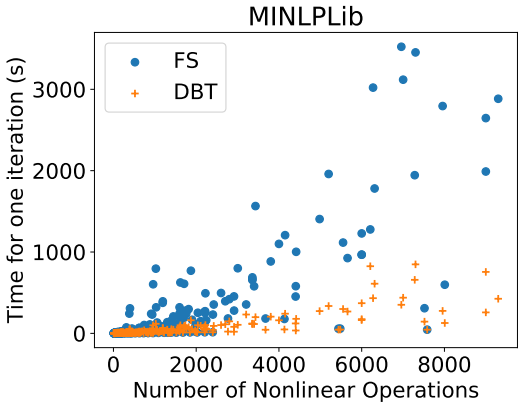

Figure 7: Comparison of the time required for the FS and DBT algorithms for problems from MINLPLib with the y-axis limited to 5000 seconds.

the bottom show results for the problems from MINLPLib. The figures on the left show the time required to complete a single iteration of bounds tightening for both the FS and DBT algorithms as a function of the number of nonlinear operations in the problem. Figure 7 is the same plot as the bottom left figure of Figure 6 except that test problems for which the FS approach takes longer than 5,000 seconds are excluded (in order to see the majority of the plot more clearly). The figures on the right of Figure 6 show, on the y-axis, the time required for one iteration by the FS algorithm divided by the time required for one iteration of the DBT algorithm. The x-axis is the time required for a single iteration of the FS algorithm. All of these figures show that the DBT algorithm scales far more favorably with problem size than the FS algorithm. Typically, the larger the problem, the longer the FS algorithm takes, and the more valuable decomposition is. Additionally, the data points in the figures on the right are colored according the the percent of variables filtered by the algorithm proposed in [14]. If a variable is filtered, it is proven that the variable's bounds cannot be tightened with the current relaxation, so no optimization problem needs to be solved for that variable. The figure in the bottom right of Figure 6 shows a small group of test problems for which no performance improvement is achieved. For the majority of these test problems, all or nearly all of the variables are filtered, and the FS algorithm is relatively fast because so few OBBT problems need solved.

The results presented above only consider the time required to perform BT and not how effective the algorithms are at tightening variable bounds. Figures 8, 9, and 10 do account for the effectiveness. These figures show the percent of test problems which achieve a given percentage improvement in the average, scaled variable ranges. The scaled range for a variable is given by

$$r_i = \frac{\overline{x_i} - \underline{x_i}}{\overline{x_i}^0 - \underline{x_i}^0} \tag{26}$$

where $\underline{x_i}^0$ and $\overline{x_i}^0$ are the variable bounds after feasibility based bounds tightening (FBBT) has been performed but before any OBBT has been performed. The average, scaled variable

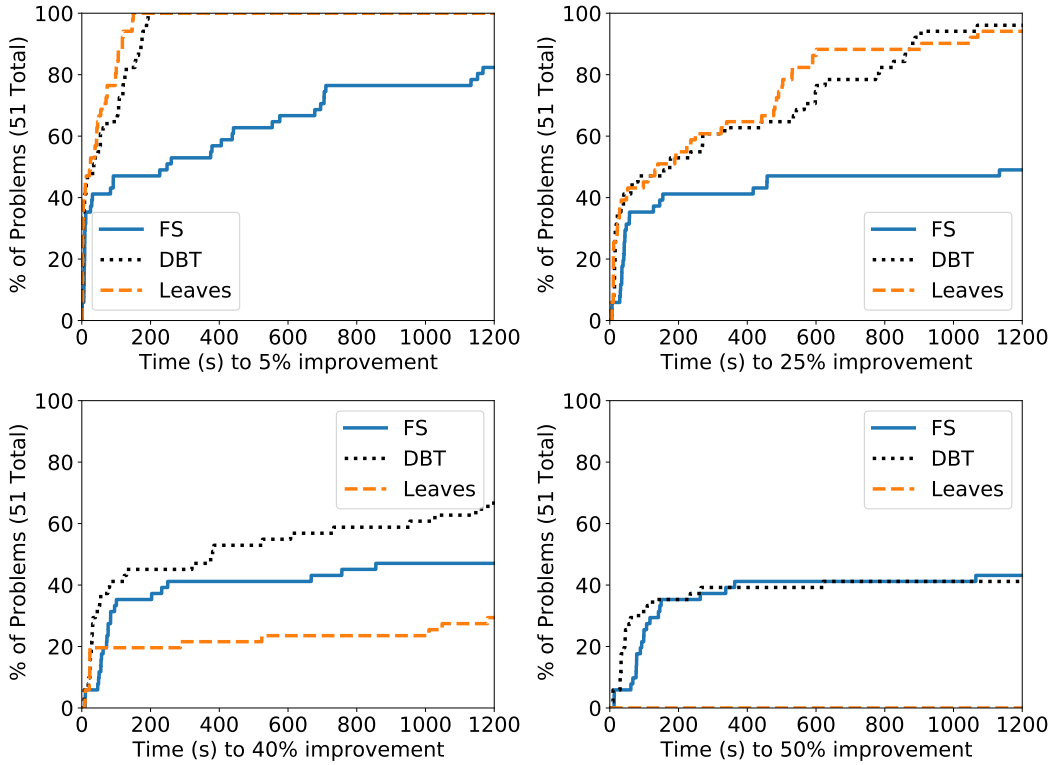

Figure 8: Performance results for the ACOPF test problems. Each figure shows the percent of test problems which achieve a given percent improvement in the average, scaled variable ranges as a function of time. From top left to bottom right, the figures show the percent of test problems which have achieved 5%, 25%, 40%, and 50% improvement in the average, scaled variable ranges.

ranges are then given by

$$\hat{r} = \frac{1}{|\mathcal{V}^{NC}|} \sum_{i \in \mathcal{V}^{NC}} r_i \tag{27}$$

Note that the initial value of $\hat{r}$ before any OBBT has been performed is 1. The percent improvement in the average, scaled variable ranges is then given by

$$100 - 100\hat{r} \tag{28}$$

We compute $\hat{r}$ after each iteration of each algorithm and use linear interpolation to find the time to a given percentage improvement. Figures 8 and 9 show results for the ACOPF test problems and test problems from MINLPLib, respectively. Figure 10 also shows results for test problems from MINLPLib, but only for test problems with at least 300 nonlinear operations. The purpose of Figure 10 is to highlight the advantage of decomposition for large problems. Note that all of the ACOPF test problems have at least 250 nonlinear operations and only 3 have less than 300 nonlinear operations.

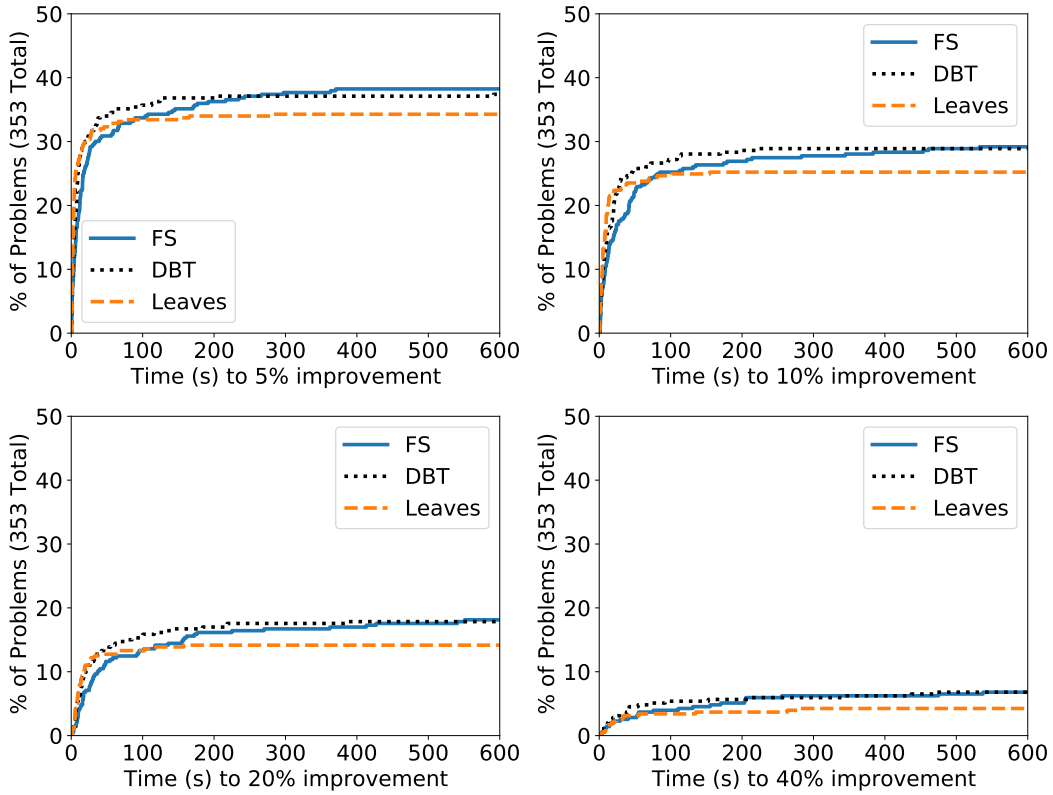

Figure 9: Performance results for the MINLPLib test problems. Each figure shows the percent of test problems which achieve a given percent improvement in the average, scaled variable ranges as a function of time. From top left to bottom right, the figures show the percent of test problems which have achieved 5%, 10%, 20%, and 40% improvement in the average, scaled variable ranges.

The top left figure of Figure 8 shows that, initially, for the ACOPF test problems, both the DBT and Leaves algorithms perform similarly, and both perform far better than the FS algorithm. This is also true for the time to 25% improvement in variable ranges. However, the Leaves algorithm achieves 40% improvement in variable ranges for very few test problems. This illustrates the value of tightening the coupling variables introduced from partitioning. Considering the time to 40% improvement in variable ranges, the DBT algorithm dominates both the Leaves and FS algorithms. Similarly, Figure 10 shows that the DBT algorithm significantly outperforms both the FS algorithm and the Leaves algorithm for the test problems from MINLPLib.

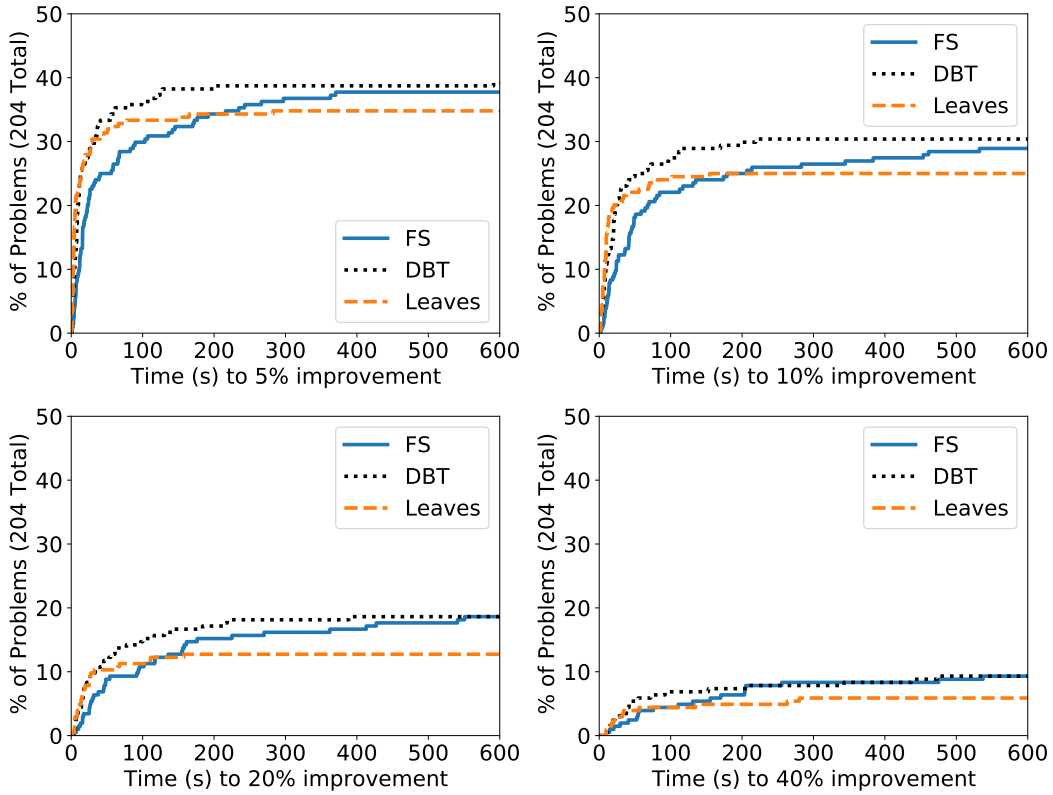

Figure 10: Performance results for the test problems from MINLPLib with at least 300 nonlinear operations. Each figure shows the percent of test problems which achieve a given percent improvement in the average, scaled variable ranges as a function of time. From top left to bottom right, the figures show the percent of test problems which have achieved 5%, 10%, 20%, and 40% improvement in the average, scaled variable ranges.

5 Conclusions

We presented two novel decomposition algorithms for performing domain reduction for non-convex optimization problems. The decomposed bounds tightening (DBT) algorithm decomposes traditional optimization-based bounds tightening (OBBT) problems by forming a bipartite graph representation of the variables and constraints in a problem and then performs graph partitioning. We recursively partition the resulting graphs, forming a binary tree of graphs, until the leaf graphs are sufficiently small. The DBT algorithm then solves a few large OBBT problems to tighten the coupling variables introduced by graph partitioning and many small OBBT problems to tighten the nonlinear and/or integer variables. The small OBBT problems are solved while only considering a small fraction of the constraints from the original problem - those contained in the corresponding graph. The Leaves algorithm is similar, except that the coupling variables are not tightened.

We compared the two proposed decomposition-based bounds tightening algorithms to traditional, full-space (FS) OBBT. Our results on a set of optimal power flow test problems and a set of test problems from MINLPLib demonstrate that the DBT algorithm drastically

outperforms the FS approach, especially for large problems. Our results also indicate that tightening the coupling variables introduced during the graph partitioning phase is critical, and the DBT algorithm also outperforms the Leaves algorithm. For large problems, the DBT algorithm can be over an order of magnitude faster than traditional OBBT and nearly as effective.

The efficacy of the DBT algorithm is encouraging, and it raises several future research questions. Existing MINLP solvers only use OBBT very selectively due to its high computational expense [10]. The DBT algorithm may be efficient enough that it can be utilized more heavily within the branch and bound tree, especially because the algorithm can be parallelized. The DBT algorithm may also complement other recent advances in domain reduction. For example, the DBT algorithm could likely complement the Lagrangian Variable Bounds proposed by Gleixner et al. [14]. Additionally, although the DBT algorithm can be parallelized, load balancing will be far more difficult compared to the standard FS approach. The best way to implement a parallel DBT algorithm remains an open question. Finally, because the DBT algorithm is so efficient, effective, and parallelizable, it may open the door to new global optimization algorithms.

6 Data Availability

The datasets analysed during the current study are available from the Power Grid Lib - Optimal Power Flow (PGLib-OPF) repository (https://github.com/power-grid-lib/pglib-opf) [3] and from MINLPLib (http://www.minlplib.org). The algorithms utilized in this paper are available in Coramin (https://github.com/coramin/coramin), including example usage. The scripts used to apply the algorithms to the datasets and generate the figures presented in this paper are available from the corresponding author on reasonable request.

7 Acknowledgements

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DENA0003525. This research was funded in part by Sandia National Laboratories' Laboratory Directed Research and Development (LDRD) Program. This work was conducted as part of the Institute for the Design of Advanced Energy Systems (IDAES) with funding from the Office of Fossil Energy, Cross-Cutting Research, U.S. Department of Energy.

The work performed by B. Knueven was done while he was a staff member at Sandia National Laboratories.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service

by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

Disclaimer: This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

References

[1] Claire S Adjiman, Stefan Dallwig, Christodoulos A Floudas, and Arnold Neumaier. A global optimization method, αbb, for general twice-differentiable constrained nlps—i. theoretical advances. *Computers & Chemical Engineering*, 22(9):1137–1158, 1998.

[2] Ioannis P Androulakis, Costas D Maranas, and Christodoulos A Floudas. αbb: A global optimization method for general constrained nonconvex problems. *Journal of Global Optimization*, 7(4):337–363, 1995.

[3] Sogol Babaeinejadsarookolaee, Adam Birchfield, Richard D Christie, Carleton Coffrin, Christopher DeMarco, Ruisheng Diao, Michael Ferris, Stephane Fliscounakis, Scott Greene, Renke Huang, et al. The power grid library for benchmarking ac optimal power flow algorithms. *arXiv preprint arXiv:1908.02788*, 2019.

[4] Pietro Belotti, Sonia Cafieri, Jon Lee, and Leo Liberti. Feasibility-based bounds tightening via fixed points. In *International Conference on Combinatorial Optimization and Applications*, pages 65–76. Springer, 2010.

[5] Pietro Belotti, Christian Kirches, Sven Leyffer, Jeff Linderoth, James Luedtke, and Ashutosh Mahajan. Mixed-integer nonlinear optimization. *Acta Numerica*, 22:1–131, 2013.

[6] Michael R Bussieck, Arne Stolbjerg Drud, and Alexander Meeraus. Minlplib—a collection of test models for mixed-integer nonlinear programming. *INFORMS Journal on Computing*, 15(1):114–119, 2003.

[7] Michael Bynum, Anya Castillo, Jean-Paul Watson, and Carl D Laird. Strengthened socp relaxations for acopf with mccormick envelopes and bounds tightening. In *Computer Aided Chemical Engineering*, volume 44, pages 1555–1560. Elsevier, 2018.

[8] Michael Bynum, Anya Castillo, Jean-Paul Watson, and Carl D Laird. Tightening mccormick relaxations toward global solution of the acopf problem. *IEEE Transactions on Power Systems*, 34(1):814–817, 2018.

[9] Francesco Ceccon, John D Siirola, and Ruth Misener. Suspect: Minlp special structure detector for pyomo. *Optimization Letters*, 14(4):801–814, 2020.

[10] Francesco Ceccon, Radu Baltean-Lugojan, Michael Bynum, Chun Li, and Ruth Misener. Galini: An extensible mixed-integer quadratically-constrained optimization solver. *Optimization Online*, 2021. URL http://www.optimization-online.org/DB_HTML/2021/01/8207.html.

[11] Chen Chen, Alper Atamtürk, and Shmuel S Oren. Bound tightening for the alternating current optimal power flow problem. *IEEE Transactions on Power Systems*, 31(5): 3729–3736, 2015.

[12] Carleton Coffrin, Hassan L Hijazi, and Pascal Van Hentenryck. Strengthening convex relaxations with bound tightening for power network optimization. In *International Conference on Principles and Practice of Constraint Programming*, pages 39–57. Springer, 2015.

[13] Robert J Dakin. A tree-search algorithm for mixed integer programming problems. *The computer journal*, 8(3):250–255, 1965.

[14] Ambros M Gleixner, Timo Berthold, Benjamin Müller, and Stefan Weltge. Three enhancements for optimization-based bound tightening. *Journal of Global Optimization*, 67(4):731–757, 2017.

[15] Chrysanthos E Gounaris, Ruth Misener, and Christodoulos A Floudas. Computational comparison of piecewise- linear relaxations for pooling problems. *Industrial & Engineering Chemistry Research*, 48(12):5742–5766, 2009.

[16] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2020. URL http://www.gurobi.com.

[17] William E Hart, Jean-Paul Watson, and David L Woodruff. Pyomo: modeling and solving mathematical programs in python. *Mathematical Programming Computation*, 3(3):219–260, 2011.

[18] William E. Hart, Carl D. Laird, Jean-Paul Watson, David L. Woodruff, Gabriel A. Hackebeil, Bethany L. Nicholson, and John D. Siirola. *Pyomo–optimization modeling in python*, volume 67. Springer Science & Business Media, second edition, 2017.

[19] MM Faruque Hasan. An edge-concave underestimator for the global optimization of twice-differentiable nonconvex problems. *Journal of Global Optimization*, 71(4):735–752, 2018.

[20] Hassan Hijazi, Carleton Coffrin, and Pascal Van Hentenryck. Convex quadratic relaxations for mixed-integer nonlinear programs in power systems. *Mathematical Programming Computation*, 9(3):321–367, 2017.

[21] George Karypis and Vipin Kumar. Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. *Retrieved from the University of Minnesota Digital Conservancy, http://hdl.handle.net/11299/215346*, 1997.

[22] Burak Kocuk, Santanu S Dey, and X Andy Sun. Strong socp relaxations for the optimal power flow problem. *Operations Research*, 64(6):1177–1196, 2016.

[23] Burak Kocuk, Santanu S Dey, and X Andy Sun. Matrix minor reformulation and socp-based spatial branch-and-cut method for the ac optimal power flow problem. *Mathematical Programming Computation*, 10(4):557–596, 2018.

[24] A.H. Land and A.G. Doig. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, pages 497–520, 1960.

[25] Jianfeng Liu, Michael Bynum, Anya Castillo, Jean-Paul Watson, and Carl D Laird. A multitree approach for global solution of acopf problems using piecewise outer approximations. *Computers & Chemical Engineering*, 114:145–157, 2018.

[26] Jianfeng Liu, Carl D Laird, Joseph K Scott, Jean-Paul Watson, and Anya Castillo. Global solution strategies for the network-constrained unit commitment problem with ac transmission constraints. *IEEE Transactions on Power Systems*, 34(2):1139–1150, 2018.

[27] Mowen Lu, Harsha Nagarajan, Russell Bent, Sandra D Eksioglu, and Scott J Mason. Tight piecewise convex relaxations for global optimization of optimal power flow. In *2018 Power Systems Computation Conference (PSCC)*, pages 1–7. IEEE, 2018.

[28] Garth P McCormick. Computability of global solutions to factorable nonconvex programs: Part i—convex underestimating problems. *Mathematical programming*, 10(1): 147–175, 1976.

[29] Ruth Misener and Christodoulos A Floudas. Antigone: algorithms for continuous/integer global optimization of nonlinear equations. *Journal of Global Optimization*, 59(2-3):503–526, 2014.

[30] Ramon E Moore, R Baker Kearfott, and Michael J Cloud. *Introduction to interval analysis*, volume 110. Siam, 2009.

[31] Yash Puranik and Nikolaos V Sahinidis. Domain reduction techniques for global nlp and minlp optimization. *Constraints*, 22(3):338–376, 2017.

[32] Hong S Ryoo and Nikolaos V Sahinidis. Global optimization of nonconvex nlps and minlps with applications in process design. *Computers & Chemical Engineering*, 19(5): 551–566, 1995.

[33] Nikolaos V Sahinidis. Baron: A general purpose global optimization software package. *Journal of global optimization*, 8(2):201–205, 1996.

[34] Artur M Schweidtmann and Alexander Mitsos. Deterministic global optimization with artificial neural networks embedded. *Journal of Optimization Theory and Applications*, 180(3):925–948, 2019.

[35] Mohit Tawarmalani and Nikolaos V Sahinidis. Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Mathematical programming*, 99(3):563–591, 2004.

[36] Angelos Tsoukalas and Alexander Mitsos. Multivariate mccormick relaxations. *Journal of Global Optimization*, 59(2-3):633–662, 2014.

[37] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57, 2006.

[38] ME Wilhelm and MD Stuber. Eago. jl: easy advanced global optimization in julia. *Optimization Methods and Software*, pages 1–26, 2020.