Strong Partitioning and a Machine Learning Approximation for Accelerating the Global Optimization of Nonconvex QCQPs

Rohit Kannan[1], Harsha Nagarajan[2], and Deepjyoti Deka[2]

[1]Grado Department of Industrial and Systems Engineering, Virginia Tech, Blacksburg, VA, USA.
[2]Applied Mathematics & Plasma Physics (T-5), Los Alamos National Laboratory, Los Alamos, NM, USA.
E-mail: {rohitkannan@vt.edu, harsha@lanl.gov, deepjyoti@lanl.gov}

Version 3 (this document): September 15, 2024

Version 2: February 22, 2023

Version 1: December 31, 2022

Abstract

We learn optimal instance-specific heuristics for the global minimization of nonconvex quadratically-constrained quadratic programs (QCQPs). Specifically, we consider partitioning-based convex mixed-integer programming relaxations for nonconvex QCQPs and propose the novel problem of strong partitioning to optimally partition variable domains *without* sacrificing global optimality. Since solving this max-min strong partitioning problem exactly can be very challenging, we design a local optimization method that leverages generalized gradients of the value function of its inner-minimization problem. However, even solving the strong partitioning problem to local optimality can be time-consuming. To address this, we propose a simple and practical machine learning (ML) approximation for homogeneous families of QCQPs. We conduct a detailed computational study on randomly generated QCQP families, including instances of the pooling problem, using the open-source global solver Alpine. Numerical experiments demonstrate that our ML approximation of strong partitioning reduces Alpine's solution time by a factor of 2 to 4.5 *on average*, with a maximum reduction factor of 10 to 200 across the different QCQP families.

Key words: Nonconvex Quadratically-Constrained Quadratic Programming, Global Optimization, Piecewise McCormick Relaxations, Strong Partitioning, Sensitivity Analysis, Machine Learning, Pooling Problem

1 Introduction

Many real-world applications involve the repeated solution of the same underlying quadratically-constrained quadratic program (QCQP) with slightly varying model parameters. Examples include the pooling problem with varying input qualities [44] and the cost-efficient operation of the power grid with varying loads and renewable sources [9]. These hard optimization problems are typically solved using off-the-shelf global optimization software [8, 45, 47, 54] that do not fully exploit the shared problem structure—heuristics within these implementations are engineered to work well *on average* over a *diverse* set of instances and may perform suboptimally for instances from a specific application [38]. Recent work [6, 39] has shown that tailoring branching decisions can significantly accelerate branch-and-bound (B&B) algorithms for mixed-integer linear programs (MILPs). In contrast, only a few papers (see Section 2.2) attempt to use machine learning (ML) to accelerate the guaranteed global solution of nonconvex *nonlinear* programs.

We use ML to accelerate partitioning algorithms [7, 55, 60] for the global minimization of nonconvex QCQPs. Partitioning algorithms determine lower bounds on the optimal value of a nonconvex QCQP using piecewise convex relaxations. They begin by selecting a subset of the continuous variables involved in nonconvex terms for partitioning. At each iteration, they refine the partitions of these variables' domains and update the piecewise convex relaxations in their lower bounding formulation. A convex mixed-integer program (MIP) is then solved to determine a lower bound [40, 59]. Partitioning algorithms typically use heuristics to specify the locations of partitioning points and continue to refine their variable partitions until the lower bounds converge to the optimal objective value. Since the complexity of their MIP relaxations can grow significantly with each iteration, the choice of partitioning points in the initial iterations can have a huge impact on the performance of these algorithms [47]. Despite their importance, the optimal selection of partitioning points is not well understood, and current approaches resort to heuristics such as bisecting the active partition [15, 55, 60] or adding partitioning points near the lower bounding solution [7, 47] to refine variable partitions.

Proposed approach. We learn to optimally partition variables' domains in a given QCQP *without* sacrificing global optimality. Similar to strong branching in B&B algorithms for MIPs [1, 44], we introduce *strong partitioning*, a new concept in the global optimization literature, to select partitioning points for the construction of piecewise convex relaxations of nonconvex problems. The key idea behind strong partitioning is to determine a specified number of partitioning points per variable such that the resulting piecewise convex relaxation-based lower bound is maximized. We formulate strong partitioning as a max-min problem, where the outer-maximization selects the partitioning points and the inner-minimization solves the piecewise convex relaxation-based lower bounding problem for a given partition. Since solving this max-min problem exactly can be very challenging, we design a local optimization method that uses generalized gradients of the value function of the inner-minimization problem within a bundle solver for nonsmooth nonconvex optimization. However, even finding a local solution to this max-min problem can be computationally expensive as each iteration of the bundle method requires the solution of a MIP. Therefore, we propose a simple and practical off-the-shelf ML model to imitate the strong partitioning strategy for homogeneous QCQP instances. We evaluate the performance of strong partitioning and its ML approximation on randomly generated QCQP instances, including instances of the pooling problem, using the open-source global solver Alpine [46, 47]. Numerical experiments demonstrate that our ML approximation of strong partitioning reduces Alpine's solution time by a factor of 2 to 4.5 on average, with a maximum reduction factor of 10 to 200 across the different QCQP families. Additionally, the results indicate that if an efficient ML model could perfectly imitate the strong partitioning strategy, it would reduce Alpine's solution time by a factor of 3.5 to 16.5 on average and by a maximum factor of 15 to 700 over the same set of instances. This underscores the potential of strong partitioning as an expert strategy for learning to accelerate the global optimization of nonconvex QCQPs.

This paper is organized as follows. Section 2 reviews ML approaches to accelerate the *guaranteed global solution* of MILPs and mixed-integer nonlinear programs. Section 3 outlines partitioning-based bounding methods for nonconvex QCQPs. Section 4 introduces strong partitioning and designs an algorithm for its local solution with theoretical guarantees. Section 5 outlines our ML approximation of strong partitioning for homogeneous QCQP families, and Section 6 presents detailed computational results demonstrating the effectiveness of using strong partitioning and our ML approximation to select Alpine's partitioning points for randomly generated families of QCQPs. We conclude with directions for future research in Section 7.

Notation. Let $[n] := \{1, \dots, n\}$, $\mathbb{R}_+$ denote the set of nonnegative reals, $\mathcal{S}^n$ denote the set of symmetric $n \times n$ matrices, and $\langle A, B \rangle$ denote the Frobenius inner product between $A, B \in \mathcal{S}^n$. We write v_i to denote the ith component of $v = (v_1, v_2, \dots, v_n) \in \mathbb{R}^n$, M_i and M_{ij} to denote the ith row and (i, j)th entry of a matrix M, and $|\mathcal{C}|$, $\text{ri}(\mathcal{C})$, and $\text{conv}(\mathcal{C})$ to denote the cardinality, relative interior, and convex hull of a set $\mathcal{C}$.

2 Related work

Optimization solvers tune algorithmic parameters through extensive testing on benchmark libraries. Many solvers also incorporate problem-specific analysis to set key parameters. However, these tunings are typically based on efficiently computable heuristics designed to perform well across a broad range of instances, which may not fully exploit the unique characteristics of each specific instance. Machine learning offers the potential to efficiently approximate more sophisticated, instance-specific heuristics, and could lead to significant computational gains for particularly challenging instances. Bengio et al. [6], Cappart et al. [14] and Lodi and Zarpellon [39] survey the burgeoning field of using ML to accelerate MILP and combinatorial optimization algorithms. In the next sections, we review related approaches on learning to branch for MILPs and learning to accelerate the guaranteed solution of (mixed-integer) nonlinear programs.

2.1 Learning to branch for MILPs

Branch-and-bound and its variants form the backbone of modern MILP solvers. The choice of branching variable at each node of the B&B tree can have a huge impact on the run time of B&B algorithms [1]. Strong branching is a heuristic for selecting the branching variable that often empirically results in a small number of B&B nodes explored. It selects the variable that maximizes the product of improvements in the lower bounds of the two child nodes, assuming both are feasible. While strong branching results in a 65% reduction in the number of nodes explored by the B&B tree on average (relative to the default branching strategy) over standard test instances, it also leads to a 44% increase in the average solution time [1]. MILP solvers therefore tend to use computationally cheaper heuristic approximations of strong branching, such as reliability, pseudocost, or hybrid branching. Motivated by the promise of strong branching, most approaches on learning to branch for MILPs aim to develop a computationally efficient and effective ML approximation, e.g., using extremely randomized trees [3], support vector machines [33], or graph neural networks [23, 48]. Other ML approaches for branching variable selection use online and reinforcement learning [27], or learn combinations of existing heuristics to make better branching decisions [18].

2.2 Learning to solve mixed-integer nonlinear problems

There are relatively fewer approaches in the literature for accelerating the *guaranteed global solution* of nonconvex (mixed-integer) nonlinear programs using ML. To the best of our knowledge, none of these approaches use ML to accelerate partitioning-based global optimization algorithms.

Baltean-Lugojan et al. [4] consider the global solution of nonconvex QCQPs using semidefinite programming relaxations. To mitigate the computational burden of these relaxations, they use ML to construct effective linear outer-approximations. Specifically, they train a neural network to select cuts from a semidefinite relaxation based on their sparsity and predicted impact on the objective. Their approach results in computationally efficient relaxations that can be effectively integrated into global solvers.

Ghaddar et al. [24] explore branching variable selection in a B&B search tree embedded within the reformulation-linearization technique for solving polynomial problems [56]. They use ML to choose the "best branching strategy" from a portfolio of branching rules, designing several hand-crafted features to optimize a quantile regression forest-based approximation of their performance metric. González-Rodríguez et al. [25] build on this approach by using ML to select a subset of second-order cone and semidefinite constraints and further strengthen the formulation.

Bonami et al. [11] train classifiers to predict whether linearizing products of binary variables or binary and continuous variables is computationally advantageous for solving mixed-integer quadratic programs. Nannicini et al. [49] train a support vector machine classifier to decide if an expensive optimality-based bound tightening routine should replace a cheaper feasibility-based routine for mixed-integer nonlinear programs. Cengil et al. [16] consider the AC optimal power flow problem and train a neural network to identify a small subset of lines and buses for which an optimality-based bound tightening routine is applied. Finally, Lee et al. [36] use classification and regression techniques to identify effective cuts for the generalized Benders decomposition master problem.

In the next section, we review partitioning algorithms for the global minimization of QCQPs, before introducing strong partitioning and an ML approximation to accelerate these algorithms.

3 Partitioning-based bounds for QCQPs

Consider the nonconvex QCQP

$$\begin{aligned} \min_{x\in[0,1]^n} \quad & x^{\mathrm{T}}Q^0x + (r^0)^{\mathrm{T}}x \\ \text{s.t.} \quad & x^{\mathrm{T}}Q^ix + (r^i)^{\mathrm{T}}x \le b_i, \quad \forall i \in [m], \end{aligned} \tag{1}$$

where $b \in \mathbb{R}^m$, $r^i \in \mathbb{R}^n$, $\forall i \in \{0\} \cup [m]$, and $Q^i \in \mathcal{S}^n$, $\forall i \in \{0\} \cup [m]$, are not assumed to be positive semidefinite. QCQPs with equality constraints and different variable bounds can be handled using simple transformations. Polynomial optimization problems may also be reformulated as QCQPs through the addition of variables and constraints. Pardalos and Vavasis [52] show that the special case of problem (1) where Q^0 has a single negative eigenvalue and $Q^i = 0$, $\forall i \in [m]$, is NP-hard.

QCQPs arise in several applications [22, 44] such as facility location [35], refinery optimization [30, 61], and electric grid optimization [9]. By introducing auxiliary variables and constraints, we can reformulate problem (1) into the following equivalent form:

$$\begin{aligned} v^* := \min_{x\in[0,1]^n, W\in\mathcal{S}^n} \quad & (r^0)^{\mathrm{T}}x + \langle Q^0, W\rangle \\ \text{s.t.} \quad & (r^i)^{\mathrm{T}}x + \langle Q^i, W\rangle \le b_i, \quad \forall i \in [m], \\ & W_{ij} = x_ix_j, \quad \forall (i,j) \in \mathcal{B}, \\ & W_{kk} = x_k^2, \quad \forall k \in \mathcal{Q}, \end{aligned} \tag{QCQP}$$

where the index sets $\mathcal{B} \subset \{(i,j) \in [n]^2 : i < j\}$ and $\mathcal{Q} \subset [n]$ denote the (pairs of) variables participating in distinct bilinear and univariate quadratic terms. We assume for simplicity that (QCQP) is feasible, and define $\mathcal{F} := \{(x,W) \in [0,1]^n \times \mathcal{S}^n : (r^i)^{\mathrm{T}}x + \langle Q^i, W\rangle \le b_i,\ \forall i \in [m]\}$ for convenience.

Al-Khayyal and Falk [2] propose to use termwise McCormick relaxations [43] to construct lower bounds on the optimal value of (QCQP). Specifically, they employ the following convex lower bounding problem within a spatial B&B framework for solving (QCQP) to global optimality:

$$\begin{aligned} \min_{(x,W)\in\mathcal{F}} \quad & (r^0)^{\mathrm{T}}x + \langle Q^0, W\rangle \\ \text{s.t.} \quad & \max\{0, x_i + x_j - 1\} \le W_{ij} \le \min\{x_i, x_j\}, \quad \forall (i,j) \in \mathcal{B}, \\ & x_k^2 \le W_{kk} \le x_k, \quad \forall k \in \mathcal{Q}. \end{aligned} \tag{2}$$

Several papers [e.g., 5, 7, 10, 13, 50, 51, 55–57, 60] improve upon this termwise McCormick bound, usually at an increase in the computational cost but with the goal of reducing the overall time for the B&B algorithm to converge. In this work, we use piecewise McCormick relaxations [7, 15, 47, 55, 60] to iteratively strengthen the lower bounding problem (2).

Piecewise McCormick relaxations begin by partitioning the domains of (a subset of) variables participating in nonconvex terms into subintervals. Let

$$\mathcal{NC} := \big\{i \in [n] : \exists j \in [n] \text{ such that } (i,j) \in \mathcal{B} \text{ or } (j,i) \in \mathcal{B}\big\} \cup \mathcal{Q}$$

denote the set of indices of variables participating in nonconvex terms within (QCQP). We assume, without loss of generality, that $\mathcal{NC} = \{1, 2, \ldots, |\mathcal{NC}|\}$, i.e., only the first $|\mathcal{NC}|$ variables x_i participate in nonconvex terms. Furthermore, we assume for simplicity that the domain of each variable x_i, $i \in \mathcal{NC}$, is partitioned

into $d+1$ subintervals, where $d \geq 1$ (other partitioning schemes can be handled similarly). Let P denote the $|\mathcal{NC}| \times (d+2)$ matrix of partitioning points (including the variable bounds 0 and 1), where

$$P_i := (P_{i1}, P_{i2}, \ldots, P_{i(d+1)}, P_{i(d+2)}), \quad \text{with} \quad 0 =: P_{i1} \leq P_{i2} \leq \cdots \leq P_{i(d+1)} \leq P_{i(d+2)} := 1, \quad \forall i \in \mathcal{NC},$$

denotes the vector of $d+2$ partitioning points for variable x_i, $i \in \mathcal{NC}$. Unlike approaches that select a subset of variables x_i, $i \in \mathcal{NC}$, involved in nonconvex terms for partitioning [47], we partition the domains of *all* such variables. While partitioning only a subset may suffice to guarantee convergence, the resulting partitioning algorithms for (QCQP) may suffer from the cluster problem in reduced-space global optimization [31, 32], potentially requiring significantly more iterations.

The piecewise McCormick relaxation-based lower bounding problem for (QCQP) can be expressed as:

$$\begin{aligned} \min_{(x,W)\in\mathcal{F}} \quad & (r^0)^{\mathrm{T}} x + \langle Q^0, W \rangle \\ \text{s.t.} \quad & (x_i, x_j, W_{ij}) \in \mathcal{PMR}^B_{ij}(P_i, P_j), \quad \forall (i,j) \in \mathcal{B}, \\ & (x_k, W_{kk}) \in \mathcal{PMR}^Q_k(P_k), \quad \forall k \in \mathcal{Q}, \end{aligned} \tag{3}$$

where $\mathcal{PMR}^B_{ij}(P_i, P_j)$ and $\mathcal{PMR}^Q_k(P_k)$ denote the feasible regions of the piecewise McCormick relaxations of $W_{ij} = x_i x_j$ and $W_{kk} = x_k^2$, respectively, obtained using the partitioning matrix P. While there are several ways of formulating these piecewise McCormick relaxations, we use the "convex combination" or "lambda" formulation below [see 34, for enhancements in the multilinear setting].

The piecewise McCormick relaxation for the constraint $W_{ij} = x_i x_j$ is written as [59]:

$$\mathcal{PMR}^B_{ij}(P_i, P_j) := \Big\{ (x_i, x_j, W_{ij}) \;:\; \exists \Lambda^{ij} \in \mathbb{R}_+^{(d+2)\times(d+2)},\, Y_i \in \{0,1\}^{(d+1)},\, Y_j \in \{0,1\}^{(d+1)} \\ \text{s.t. } (x_i, x_j, W_{ij}, \Lambda^{ij}, Y_i, Y_j) \text{ satisfies } (4a) - (4d) \Big\},$$

where

$$x_i = \sum_{k,l=1}^{d+2} \Lambda^{ij}_{kl} P_{il}, \qquad x_j = \sum_{k,l=1}^{d+2} \Lambda^{ij}_{kl} P_{jk}, \qquad W_{ij} = \sum_{k,l=1}^{d+2} \Lambda^{ij}_{kl} P_{il} P_{jk}, \tag{4a}$$

$$\sum_{l=1}^{d+1} Y_{il} = 1, \qquad \sum_{k=1}^{d+1} Y_{jk} = 1, \qquad \sum_{k,l=1}^{d+2} \Lambda^{ij}_{kl} = 1, \tag{4b}$$

$$\sum_{k=1}^{d+2} \Lambda^{ij}_{k1} \leq Y_{i1}, \qquad \sum_{k=1}^{d+2} \Lambda^{ij}_{k(d+2)} \leq Y_{i(d+1)}, \qquad \sum_{k=1}^{d+2} \Lambda^{ij}_{k(l+1)} \leq Y_{il} + Y_{i(l+1)}, \;\; \forall l \in [d], \tag{4c}$$

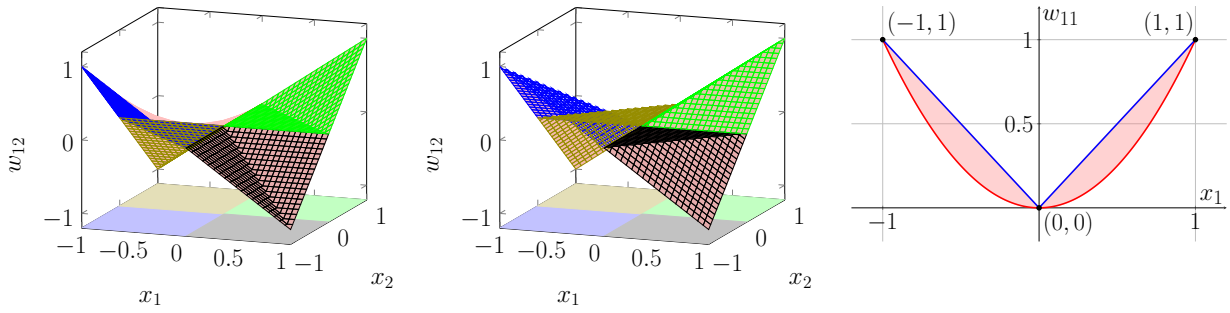
$$\sum_{l=1}^{d+2} \Lambda^{ij}_{1l} \leq Y_{j1}, \qquad \sum_{l=1}^{d+2} \Lambda^{ij}_{(d+2)l} \leq Y_{j(d+1)}, \qquad \sum_{l=1}^{d+2} \Lambda^{ij}_{(k+1)l} \leq Y_{jk} + Y_{j(k+1)}, \;\; \forall k \in [d]. \tag{4d}$$

Only equations (4a) depend on the partitioning matrix P, which is a parameter in these constraints. The binary vectors Y_i and Y_j denote the active partition of x_i and x_j—these variables are reused in the piecewise McCormick relaxations of other nonconvex terms involving x_i or x_j.

The piecewise McCormick relaxation for $W_{kk} = x_k^2$ is written as [40, 47]:

$$\mathcal{PMR}^Q_k(P_k) := \Big\{ (x_k, W_{kk}) \;:\; \exists \Lambda^k \in \mathbb{R}_+^{(d+2)},\, Y_k \in \{0,1\}^{(d+1)} \text{ s.t. } (x_k, W_{kk}, \Lambda^k, Y_k) \text{ satisfies } (5a) - (5c) \Big\},$$

Figure 1: Illustration of piecewise McCormick relaxations for a bilinear and a univariate quadratic term. The variable domains are changed from $[0,1]$ to $[-1,1]$ for better illustration. The left and middle plots illustrate the lower and upper parts, respectively, of the piecewise McCormick relaxation for the bilinear term $w_{12} = x_1 x_2$ on the domain $x_1, x_2 \in [-1,1]$ with partitions $P_1 = P_2 = (-1,0,1)$. The right plot illustrates the piecewise McCormick relaxation for the quadratic term $w_{11} = x_1^2$ on the domain $x_1 \in [-1,1]$ (the lower part coincides with the red quadratic curve) with the partition $P_1 = (-1,0,1)$.



where

$$x_k = \sum_{l=1}^{d+2} \Lambda_l^k P_{kl}, \qquad W_{kk} \le \sum_{l=1}^{d+2} \Lambda_l^k (P_{kl})^2, \qquad \sum_{l=1}^{d+1} Y_{kl} P_{kl} \le x_k \le \sum_{l=1}^{d+1} Y_{kl} P_{k(l+1)}, \tag{5a}$$

$$\sum_{l=1}^{d+1} Y_{kl} = 1, \qquad \sum_{l=1}^{d+2} \Lambda_l^k = 1, \qquad W_{kk} \ge x_k^2, \tag{5b}$$

$$\Lambda_1^k \le Y_{k1}, \qquad \Lambda_{d+2}^k \le Y_{k(d+1)}, \qquad \Lambda_{l+1}^k \le Y_{kl} + Y_{k(l+1)}, \ \forall l \in [d]. \tag{5c}$$

Only equations (5a) depend on the partitioning matrix P, which is again a parameter in these constraints. The third set of constraints in (5a) are redundant for the description of the set $\mathcal{PMR}_k^Q(P_k)$, but strengthen its convex relaxation. Note that equations (5b) involve convex quadratic functions of x_k. Figure 1 illustrates the piecewise McCormick relaxations for a bilinear and a univariate quadratic term.

Using the above representations of $\mathcal{PMR}_{ij}^B$ and $\mathcal{PMR}_k^Q$, we get the following extended *convex* mixed-integer QCQP formulation for the piecewise McCormick relaxation problem (3):

$$\begin{aligned} \min_{\substack{(x,W)\in\mathcal{F} \\ \Lambda\ge 0, Y\in\mathcal{Y}}} \ & (r^0)^{\mathrm{T}} x + \langle Q^0, W \rangle \\ \text{s.t.} \ & (x_i, x_j, W_{ij}, \Lambda^{ij}, Y_i, Y_j) \text{ satisfies (4a)} - \text{(4d)}, \quad \forall (i,j) \in \mathcal{B}, \\ & (x_k, W_{kk}, \Lambda^k, Y_k) \text{ satisfies (5a)} - \text{(5c)}, \quad \forall k \in \mathcal{Q}, \end{aligned} \tag{PMR}$$

where $Y \in \{0,1\}^{|\mathcal{NC}|\times(d+1)}$, $\mathcal{Y} := \left\{ Y \in \{0,1\}^{|\mathcal{NC}|\times(d+1)} : \sum_{l=1}^{d+1} Y_{il} = 1, \ \forall i \in \mathcal{NC} \right\}$ is an intersection of special-ordered sets of type 1, and Λ comprises Λ^{ij}, $(i,j) \in \mathcal{B}$, and Λ^k, $k \in \mathcal{Q}$. We call the jth partition $[P_{ij}, P_{i(j+1)}]$ of variable x_i, $i \in \mathcal{NC}$, active if there exists an optimal solution to (PMR) with an x-component $\bar{x} \in [0,1]^n$ such that $P_{ij} \le \bar{x}_i \le P_{i(j+1)}$. Equivalently, the jth partition $[P_{ij}, P_{i(j+1)}]$ of x_i is said to be active if there exists an optimal solution to (PMR) with a Y-component $\bar{Y} \in \mathcal{Y}$ such that $\bar{Y}_{ij} = 1$.

Algorithm 1 outlines a partitioning-based global optimization algorithm that solves problem (PMR) to determine a sequence of lower bounds on the optimal value of (QCQP). It assumes that local search can find a near-optimal solution to (QCQP) within a finite number of iterations—a standard assumption that often holds in practice. The convergence of the lower bound is typically the limiting factor in the convergence of partitioning-based algorithms. In particular, the choice of heuristic on line 7 of Algorithm 1 for refining the

Algorithm 1 Partitioning-based global optimization algorithm for (QCQP)

Input: relative optimality tolerance $\varepsilon_r > 0$.
Initialization: partitioning points $P_i^0 := (0, 1)$, $\forall i \in \mathcal{NC}$, best found solution $\{\hat{x}\} = \emptyset$ with objective $UBD = +\infty$, lower bound $LBD = -\infty$, and iteration number $l = 0$.
Solve (QCQP) locally. Update the incumbent $\hat{x}$ and upper bound UBD if relevant.
Solve the termwise McCormick relaxation (2). Update the lower bound LBD.
while $\dfrac{UBD - LBD}{|UBD| + 10^{-6}} > \varepsilon_r$ **or** $UBD = +\infty$ **do**
    Update $l \leftarrow l + 1$.
    Partition refinement: add new partitioning points to P^{l-1} to obtain the partitioning matrix P^l.
    Solve (PMR) with the partitioning matrix P^l. Update the lower bound LBD.
    Solve (QCQP) locally. Update the incumbent $\hat{x}$ and upper bound UBD if needed.
end while
Return the ε_r-optimal x-solution $\hat{x}$, upper bound UBD, and lower bound LBD.

Algorithm 2 Generic partition refinement policy for iteration l

Input: partitioning matrix P^{l-1} used to construct piecewise McCormick relaxations at iteration $l-1$, index $\mathcal{A}(i, l-1) \in \mathbb{N}$ of an active partition for variable x_i at iteration $l-1$, $\forall i \in \mathcal{NC}$, and the number of new partitioning points $d_l \in \mathbb{N}$ to be added to P_i^{l-1}, $\forall i \in \mathcal{NC}$.
Initialization: set $P^l = P^{l-1}$.
for $i \in \mathcal{NC}$ **do**
    Add d_l new partitioning points to the active partition $\left[P^{l-1}_{i(\mathcal{A}(i,l-1))}, P^{l-1}_{i(\mathcal{A}(i,l-1)+1)}\right]$ for variable x_i to obtain a refined vector of partitioning points P_i^l for x_i.
end for
Output: partitioning matrix P^l used to construct piecewise McCormick relaxations at iteration l.

partitioning matrix P can greatly impact both the number of iterations and the time required for convergence. This motivates the concept of strong partitioning, which selects the partitioning matrix P to maximize the piecewise McCormick relaxation-based lower bound. We define partition refinement policies in Section 3.1, and introduce the strong partitioning policy and an ML approximation in Sections 4.2 and 5. Section 6.4 details the partition refinement policy implemented within Alpine [47]. Note that Algorithm 1 can be readily adapted to include enhancements such as bound tightening.

3.1 Partitioning policy

Algorithm 2 outlines a generic policy for refining variable partitions at any given iteration l of Algorithm 1. It adds d_l new partitioning points to the active partition of each variable x_i involved in nonconvex terms. Various partition refinement strategies, such as bisecting the active partition [15, 55, 60], adding partitioning points near the lower bounding solution [7, 47], and the strong partitioning and ML-based policies described in Sections 4.2 and 5, can be seen as specific instantiations of Algorithm 2.

In the next two sections, we introduce the strong partitioning and ML-based policies, which differ from the partitioning policy implemented in Alpine (see Section 6.4 for details) only during the first iteration.

4 Strong partitioning for nonconvex QCQPs

The choice of partitioning points in the initial iterations can greatly impact the strength of lower bounds, the number of iterations needed for convergence, and the overall solution time. We introduce the *Strong Partitioning* (SP) policy to overcome the limitations of existing heuristics for selecting partitioning points.

Before presenting the concept of strong partitioning, we further relax (PMR) by outer-approximating the convex quadratic terms in equation (5b) to obtain the following MILP relaxation. For purely bilinear programs ($\mathcal{Q} = \emptyset$), problem (PMR) is already an MILP, and this outer-approximation step is unnecessary.

$$\begin{aligned} \underline{v}(P) := \min_{\substack{(x,W)\in\mathcal{F} \\ \Lambda\geq 0, Y\in\mathcal{Y}}} \ & (r^0)^{\mathrm{T}} x + \langle Q^0, W\rangle && \text{(PMR-OA)} \\ \text{s.t.} \ & (x_i, x_j, W_{ij}, \Lambda^{ij}, Y_i, Y_j) \text{ satisfies (4a)} - \text{(4d)}, && \forall (i,j) \in \mathcal{B}, \\ & (x_k, W_{kk}, \Lambda^k, Y_k) \text{ satisfies (5a) and (5c)}, && \forall k \in \mathcal{Q}, \\ & \sum_{l=1}^{d+2} \Lambda_l^k = 1, \quad W_{kk} \geq (2\alpha_j^k) x_k - (\alpha_j^k)^2, && \forall j \in \mathcal{J}_k, \ k \in \mathcal{Q}. \end{aligned} \tag{6}$$

We explicitly indicate the dependence of the piecewise McCormick lower bound $\underline{v}$ on the partitioning matrix P. Constraints (6) outer-approximate the inequalities $W_{kk} \geq x_k^2$ in equation (5b) at the points $\{\alpha_j^k\}_{j\in\mathcal{J}_k} \subset [0,1]$, which are assumed to include $P_{k1}, \ldots, P_{k(d+2)}$. This outer-approximation step is needed to compute a generalized gradient of the value function $\underline{v}$ with respect to the partitioning matrix P, a key component of our SP algorithm (see Section 4.1 for details). *We solve problem* (PMR-OA) *only during strong partitioning* and revert to solving problem (PMR) when computing lower bounds within Algorithm 1.

We recast (PMR-OA) into the following abstract form for mathematical convenience, using suitably defined vectors c and d, matrices $\bar{M}$ and $\bar{B}$, matrix-valued function M with co-domain $\mathbb{R}^{n_r \times n_c}$, and a vector of variables z (including variables x, W, Λ, and auxiliary/slack variables):

$$\underline{v}(P) := \min_{Y\in\mathcal{Y}} v(P,Y), \qquad \text{where} \qquad \begin{aligned} v(P,Y) := \min_{z\geq 0} \ & c^{\mathrm{T}} z \\ \text{s.t.} \ & M(P) z = d, \\ & \bar{M} z = \bar{B} \operatorname{vec}(Y), \end{aligned} \tag{PMR-OA-LP}$$

where $\operatorname{vec}(Y)$ denotes the vectorization of matrix Y. We omit in (PMR-OA-LP) the third set of constraints in equation (5a) because they are redundant for the piecewise McCormick relaxations $\mathcal{PMR}_k^Q$. The constraints $\bar{M} z = \bar{B} \operatorname{vec}(Y)$ represent equations (4c), (4d), and (5c) by adding slack variables. They ensure that for any $Y \in \mathcal{Y}$, at most four of the Λ^{ij} variables in the extended formulation of each set $\mathcal{PMR}_{ij}^B$ and at most two of the Λ^k variables in the extended formulation of each set $\mathcal{PMR}_k^Q$ may be nonzero.

The concept of strong partitioning is analogous to strong branching in B&B algorithms for MILPs. While strong branching for MILPs involves selecting the branching variable—a discrete choice—to maximize the product of improvements in the lower bounds of the two child nodes, strong partitioning selects partitioning points for *each* partitioned variable—continuous choices within the variable domains—to maximize the piecewise McCormick relaxation lower bound. It can be formulated as the following max-min problem:

$$P^* \in \arg\max_{P\in\mathcal{P}_d} \underline{v}(P), \tag{SP}$$

where $\underline{v}(P)$ is the optimal value function of problem (PMR-OA), and the set $\mathcal{P}_d$ is defined as:

$$\mathcal{P}_d := \left\{ P \in [0,1]^{|\mathcal{NC}|\times(d+2)} \ : \ 0 = P_{i1} \leq P_{i2} \leq \cdots \leq P_{i(d+1)} \leq P_{i(d+2)} = 1, \ \forall i \in \mathcal{NC} \right\}.$$

The strong partitioning problem (SP) is challenging to solve even to local optimality because the inner problem (PMR-OA) includes binary decisions and its feasible region depends on P (variables of the outer

problem). While (SP) can be formulated as a generalized semi-infinite program, current global optimization algorithms for this problem class do not scale well [29]. Therefore, we design a local optimization method for (SP) aimed at determining a partitioning matrix $\bar{P} \in \mathcal{P}_d$ that yields a tight lower bound $\underline{v}(\bar{P})$. We use this local solution of (SP) to specify the partitioning matrix P^1 at the first iteration of Algorithm 1. If the resulting lower and upper bounds LBD and UBD in Algorithm 1 do not converge after the first iteration, we use the partitioning policy implemented within Alpine (see Section 6.4) to specify the partitioning matrix P^l from its second iteration ($l \geq 2$). Section 4.2 outlines the strong partitioning policy.

We use generalized gradients of the value function of problem (PMR-OA) within a bundle solver for nonsmooth nonconvex optimization to solve problem (SP) to local optimality. Although value functions of MILPs are generally discontinuous, problem (PMR-OA) possesses special structure because outer-approximations of the piecewise McCormick relaxations $\mathcal{PMR}_{ij}^{B}(P_i, P_j)$ and $\mathcal{PMR}_{k}^{Q}(P_k)$ can be described using nonconvex piecewise-linear continuous functions (cf. Figure 1). As shown in Theorems 3 and 4 in Section 4.1, this structure allows us to compute sensitivity information for the value function $\underline{v}$.

We use the bundle solver MPBNGC, which requires function and generalized gradient evaluations at points $P \in \mathcal{P}_d$ during its algorithm [see 42, for details]. Each function evaluation $\underline{v}(P)$ necessitates solving an MILP, (PMR-OA). Under suitable assumptions, a generalized gradient $\partial \underline{v}(P)$ can be obtained by fixing variables Y to an optimal Y-solution of (PMR-OA) and computing a generalized gradient of the resulting LP, (PMR-OA-LP). We formalize these details in Section 4.1. Before proceeding, we summarize the convergence guarantees of MPBNGC [42] below for completeness.

Definition 1. Let $Z \subset \mathbb{R}^N$ be open. A locally Lipschitz function $f : Z \to \mathbb{R}$ is said to be weakly semismooth if the directional derivative $f'(z,d) = \lim_{t \downarrow 0} \frac{f(z+td)-f(z)}{t}$ exists for all $z \in Z$ and $d \in \mathbb{R}^N$, and we have $f'(z,d) = \lim_{t \downarrow 0} \xi(z+td)^{\mathrm{T}} d$ for some $\xi(z+td) \in \partial f(z+td)$.

Definition 2. Let $f : \mathbb{R}^N \to \mathbb{R}$ and $g : \mathbb{R}^N \to \mathbb{R}^M$ be locally Lipschitz continuous functions. Consider the problem $\min_{z:g(z)\leq 0} f(z)$. A feasible point z^* is said to be substationary if there exist multipliers $\lambda \geq 0$ and $\mu \in \mathbb{R}^M_+$, with $(\lambda, \mu) \neq (0,0)$, such that $0 \in \lambda \partial f(z^*) + \sum_{j=1}^{M} \mu_j \partial g_j(z^*)$ and $\mu_j g_j(z^*) = 0$, $\forall j \in [M]$.

Theorem 1. Suppose the value function $\underline{v}$ of (PMR-OA) is weakly semismooth. Then MPBNGC either terminates finitely with a substationary point to (SP), or any accumulation point of a sequence of MPBNGC solutions is a substationary point to (SP).

Proof. See Theorem 9 of Mäkelä [42]. □

The example below shows that the value function $\underline{v}$ of problem (PMR-OA) may be nonsmooth.

Example 1. Consider the following instance of the QCQP (1):

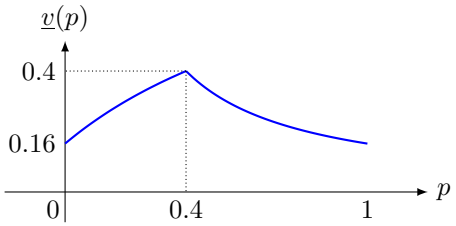
$$\min_{x \in [0,1]} \; x \quad \text{s.t.} \quad x^2 \geq (0.4)^2.$$

Its optimal solution is $x^* = 0.4$ with optimal value $v^* = 0.4$. Suppose we wish to partition the domain of x into two sub-intervals ($d = 1$). Let $P = (0, p, 1)$ denote partitioning points for x with $0 \leq p \leq 1$. After some algebraic manipulation, the outer-approximation problem (PMR-OA) can be equivalently written as:

$$\underline{v}(p) = \min_{x \in [0,1]} \; x \;\text{ s.t. }\; w \geq (0.4)^2, \;\; w \leq \max\{px, (1+p)x - p\}, \;\; w \geq 2\alpha_j x - \alpha_j^2, \; \forall j \in \mathcal{J},$$

where $\{\alpha_j\}_{j \in \mathcal{J}} \subset [0,1]$ and we write $\underline{v}(p)$ to only indicate the dependence of the value function $\underline{v}$ on the single nontrivial partitioning point p. We can derive the piecewise McCormick lower bound to be:

$$\underline{v}(p) = \begin{cases} \frac{0.16+p}{1+p}, & \text{if } 0 \le p \le 0.4 \\ \frac{0.16}{p}, & \text{if } 0.4 < p \le 1 \end{cases},$$



which shows that $\underline{v}$ is continuous and piecewise differentiable at $p = 0.4$ for this example.

4.1 Computing a generalized gradient of $\underline{v}$

We identify conditions under which a generalized gradient of the value function $\underline{v}$ can be computed in practice. (For notational simplicity, we discuss generalized gradients of $\underline{v}$ on $\mathcal{P}_d$ rather than on $\text{ri}(\mathcal{P}_d)$.) We start with the following useful result. It implies that the active partitions of the variables x_i, $i \in \mathcal{NC}$, in problem (PMR-OA) for a given value of P remain active for all partitioning matrices in a sufficiently small relative neighborhood of P. Its assumption that the Y-solution to problem (PMR-OA) is unique can be verified by adding a "no-good cut" and re-solving (PMR-OA) to check if the second-best solution for Y has a strictly greater objective than $\underline{v}(P)$.

Lemma 2. Fix $P \in \mathcal{P}_d$. Suppose problem (PMR-OA) has a unique Y-solution $Y^* \in \mathcal{Y}$ and $v(\cdot, Y^*)$ is continuous at P. Then $\underline{v}(\tilde{P}) = v(\tilde{P}, Y^*)$, $\forall \tilde{P} \in \mathcal{P}_d$ in a sufficiently small relative neighborhood of P.

Proof. Because Y^* is the unique Y-solution to (PMR-OA) at $P \in \mathcal{P}_d$, we have $v(P, Y^*) < v(P, Y)$, $\forall Y \in \mathcal{Y} \backslash \{Y^*\}$. To demonstrate that $\underline{v}(\cdot) \equiv v(\cdot, Y^*)$ in a sufficiently small relative neighborhood of P, we show that the value function $v(\cdot, Y)$ is lower semicontinuous on $\mathcal{P}_d$ for each $Y \in \mathcal{Y}$. The stated result then follows, as $v(\cdot, Y^*)$ is assumed to be continuous at P.

The set-valued mapping $P \in \mathcal{P}_d \mapsto \{z \ge 0 : M(P)z = d,\ \bar{M}z = \bar{B}\,\text{vec}(Y)\}$ is locally compact for each $Y \in \mathcal{Y}$ due to the continuity of the mapping M and the finite bounds that can be deduced for all variables in (PMR-OA). Hence, Lemma 5.3 of Still [58] implies $v(\cdot, Y)$ is lower semicontinuous on $\mathcal{P}_d$, for each $Y \in \mathcal{Y}$. □

The next result characterizes the gradient of the value function $\underline{v}$ with respect to the "unfixed" partitioning points $P_{i2}, \ldots, P_{i(d+1)}$, $i \in \mathcal{NC}$. It assumes that problem (PMR-OA-LP), with Y fixed to the Y-solution Y^* of problem (PMR-OA), has unique primal and dual optimal solutions.

Theorem 3. Suppose $P \in \mathcal{P}_d$ and problem (PMR-OA) has a unique Y-solution $Y^* \in \mathcal{Y}$. Consider problem (PMR-OA-LP) with Y fixed to Y^*. If this LP has a unique primal solution z^* and a unique dual solution π^*, then

$$\frac{\partial \underline{v}}{\partial P_{ij}}(P) = \frac{\partial v}{\partial P_{ij}}(P, Y^*) = \sum_{k=1}^{n_r} \sum_{l=1}^{n_c} \pi_k^* z_l^* \frac{\partial M_{kl}}{\partial P_{ij}}(P), \quad \forall i \in \mathcal{NC},\ j \in \{2, \ldots, d+1\}.$$

Proof. Lemma 2 implies that $\underline{v}(\cdot) \equiv v(\cdot, Y^*)$ in a sufficiently small relative neighborhood of P, provided $v(\cdot, Y^*)$ is continuous at P. Theorem 1 of Freund [20] (cf. Proposition 4.1 of De Wolf and Smeers [17] and Theorem 4.3 of Still [58]) and the fact that the function M is continuously differentiable on $\mathcal{P}_d$ together imply that $v(\cdot, Y^*)$ is continuously differentiable at P and the stated equalities hold. □

Next, we derive a formula for the Clarke generalized gradient $\partial \underline{v}(P)$ when the assumption in Theorem 3 that the LP (PMR-OA-LP), with Y fixed to Y^*, has unique primal and dual solutions does *not* hold. Note that $\partial \underline{v}(P)$, $\partial v(P, Y^*)$, and $\frac{\partial M_{kl}}{\partial P}(P)$ denote generalized gradients only with respect to the "unfixed" partitioning points $P_{i2}, \ldots, P_{i(d+1)}$, $i \in \mathcal{NC}$.

Theorem 4. Suppose $P \in \mathcal{P}_d$ and problem (PMR-OA) has a unique Y-solution $Y^* \in \mathcal{Y}$. Consider problem (PMR-OA-LP) with Y fixed to Y^*. Suppose $v(\cdot, Y^*)$ is finite and locally Lipschitz in a sufficiently small relative neighborhood of P. Then

$$\partial \underline{v}(P) = \partial v(P, Y^*) = \operatorname{conv}\left(\left\{ \sum_{k=1}^{n_r} \sum_{l=1}^{n_c} \pi_k^* z_l^* \frac{\partial M_{kl}}{\partial P}(P) \ : \ (z^*, \pi^*) \text{ is a primal-dual optimal pair for (PMR-OA-LP) with } Y \text{ fixed to } Y^* \right\} \right).$$

Proof. Lemma 2 implies that $\underline{v}(\cdot) \equiv v(\cdot, Y^*)$ in a sufficiently small relative neighborhood of P. The stated equalities hold by mirroring the proof of Theorem 5.1 of De Wolf and Smeers [17] and noting that the function M is continuously differentiable on $\mathcal{P}_d$. □

The next result ensures that $v(\cdot, Y^*)$ is locally Lipschitz in a sufficiently small relative neighborhood of $P \in \mathcal{P}_d$, provided that the matrix $M(P)$ in problem (PMR-OA-LP) has full row rank and Slater's condition holds. (See Im [28] and Assumption 5.1 of De Wolf and Smeers [17] for details.)

Lemma 5. Suppose $P \in \mathcal{P}_d$ and $\bar{Y} \in \mathcal{Y}$. Consider problem (PMR-OA-LP) with Y fixed to $\bar{Y}$. If the matrix $M(P)$ has full row rank and $d \in \operatorname{int}\big(\{M(P)z : z \geq 0, \bar{M}z = \bar{B} \operatorname{vec}(\bar{Y})\}\big)$, then $v(\cdot, \bar{Y})$ is finite and locally Lipschitz in a sufficiently small relative neighborhood of P.

Proof. See Proposition 5.3 of [17] and pages 73 to 76 of [28]. □

We now verify that the full rank assumption in Lemma 5 holds in general.

Lemma 6. The matrix $M(P)$ has full row rank for each $P \in \operatorname{ri}(\mathcal{P}_d)$.

Proof. Fix $Y \in \mathcal{Y}$. Since $P \in \operatorname{ri}(\mathcal{P}_d)$, we have $0 = P_{i1} < P_{i2} < \cdots < P_{i(d+1)} < P_{i(d+2)} = 1$, $\forall i \in \mathcal{NC}$. We show that for each $(i,j) \in \mathcal{B}$ and $k \in \mathcal{Q}$, the equality constraints in (4a)-(4b) and (5a)-(5b) have full row rank, which implies that $M(P)$ has full row rank. We ignore the inequality constraints because they are converted into equality constraints by adding unique slack variables.

We begin by focusing on the equality constraints in (4a)-(4b) involving the x, W, and Λ variables. Consider a fixed $(i,j) \in \mathcal{B}$ and any pair of indices $k, l \in [d+1]$. We can rewrite these equality constraints as follows after suppressing the terms associated with most of the Λ^{ij} variables:

$$\begin{pmatrix} -1 & 0 & 0 & P_{il} & P_{il} & P_{i(l+1)} & P_{i(l+1)} & \cdots \\ 0 & -1 & 0 & P_{jk} & P_{j(k+1)} & P_{jk} & P_{j(k+1)} & \ddots \\ 0 & 0 & -1 & P_{il}P_{jk} & P_{il}P_{j(k+1)} & P_{i(l+1)}P_{jk} & P_{i(l+1)}P_{j(k+1)} & \cdots \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & \cdots \end{pmatrix} \begin{pmatrix} x_i \\ x_j \\ W_{ij} \\ \Lambda^{ij}_{kl} \\ \Lambda^{ij}_{(k+1)l} \\ \Lambda^{ij}_{k(l+1)} \\ \Lambda^{ij}_{(k+1)(l+1)} \\ \vdots \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

We show that the fourth, fifth, sixth, and seventh columns of the matrix above are linearly independent whenever $P \in \operatorname{ri}(\mathcal{P}_d)$. Suppose, by way of contradiction, that these four columns are linearly dependent. Then, there exist scalars μ_1, μ_2, μ_3, and μ_4, not all zero, such that

$$\mu_1 \begin{pmatrix} P_{il} \\ P_{jk} \\ P_{il}P_{jk} \\ 1 \end{pmatrix} + \mu_2 \begin{pmatrix} P_{il} \\ P_{j(k+1)} \\ P_{il}P_{j(k+1)} \\ 1 \end{pmatrix} + \mu_3 \begin{pmatrix} P_{i(l+1)} \\ P_{jk} \\ P_{i(l+1)}P_{jk} \\ 1 \end{pmatrix} + \mu_4 \begin{pmatrix} P_{i(l+1)} \\ P_{j(k+1)} \\ P_{i(l+1)}P_{j(k+1)} \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

This implies the following linear equations in μ_1, μ_2, μ_3, and μ_4:

$$(\mu_1 + \mu_2)P_{il} = -(\mu_3 + \mu_4)P_{i(l+1)}, \tag{7a}$$
$$(\mu_1 + \mu_3)P_{jk} = -(\mu_2 + \mu_4)P_{j(k+1)}, \tag{7b}$$
$$P_{il}\big(P_{jk}\mu_1 + P_{j(k+1)}\mu_2\big) = -P_{i(l+1)}\big(P_{jk}\mu_3 + P_{j(k+1)}\mu_4\big), \tag{7c}$$
$$\mu_1 + \mu_2 = -(\mu_3 + \mu_4). \tag{7d}$$

Since $P_{il} < P_{i(l+1)}$ and $P_{jk} < P_{j(k+1)}$, equations (7a) and (7d) imply that $\mu_1 + \mu_2 = 0$ and $\mu_3 + \mu_4 = 0$. Similarly, equations (7b) and (7d) imply that $\mu_1 + \mu_3 = 0$ and $\mu_2 + \mu_4 = 0$. Together, these equations imply $\mu_1 = \mu_4$, $\mu_2 = \mu_3$, and $\mu_1 = -\mu_2$. Combining these with equation (7c) leads to $\mu_1 = \mu_2 = \mu_3 = \mu_4 = 0$, which is a contradiction.

Next, we focus on the equality constraints in (5a)-(5b) involving the x, W, and Λ^k variables for a fixed $k \in \mathcal{Q}$. Consider any index $l \in [d+1]$. We can rewrite these equality constraints as follows after suppressing the terms associated with most of the Λ^k variables:

$$\begin{pmatrix} -1 & P_{kl} & P_{k(l+1)} & \dots \\ 0 & 1 & 1 & \dots \end{pmatrix} \begin{pmatrix} x_k \\ \Lambda^k_l \\ \Lambda^k_{l+1} \\ \vdots \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

The second and third columns of the matrix above are linearly independent whenever $P_{kl} < P_{k(l+1)}$. □

Finally, we show that for almost every $P \in \mathcal{P}_d$ with respect to the uniform measure, problem (PMR-OA) either has a unique Y-solution (allowing us to use Theorem 3 or 4 to compute a generalized gradient of $\underline{v}$ under mild conditions), or $\underline{v}(P) = v^*$ (i.e., the optimal values of (QCQP) and (PMR-OA) are equal, implying that the partitioning matrix P is sufficient for the convergence of the lower bound), or both statements hold.

Theorem 7. At least one of the following statements holds for almost every $P \in \mathcal{P}_d$:

1. $\underline{v}(P) = v^*$,
2. Problem (PMR-OA) has a unique Y-solution.

Proof. Consider $P \in \mathcal{P}_d$, and let $\hat{Y} \in \mathcal{Y}$ and $\hat{x} \in [0,1]^n$ denote the Y and x components of an optimal solution to problem (PMR-OA). We examine the following cases:

(a) For each $k \in \mathcal{Q}$, we have $\hat{x}_k \in \{P_{k1}, P_{k2}, \dots, P_{k(d+1)}, P_{k(d+2)}\}$. Additionally, for each $(i,j) \in \mathcal{B}$, either $\hat{x}_i \in \{P_{i1}, P_{i2}, \dots, P_{i(d+1)}, P_{i(d+2)}\}$, or $\hat{x}_j \in \{P_{j1}, P_{j2}, \dots, P_{j(d+1)}, P_{j(d+2)}\}$, or both.

(b) Case (a) does not hold, i.e., there either exists at least one index $k \in \mathcal{Q}$ such that $\hat{x}_k \notin \{P_{k1}, P_{k2}, \dots, P_{k(d+1)}, P_{k(d+2)}\}$, or there exists at least one pair of indices $(i,j) \in \mathcal{B}$ such that both $\hat{x}_i \notin \{P_{i1}, P_{i2}, \dots, P_{i(d+1)}, P_{i(d+2)}\}$ and $\hat{x}_j \notin \{P_{j1}, P_{j2}, \dots, P_{j(d+1)}, P_{j(d+2)}\}$.

Suppose case (a) holds. Since we assume that $\{P_{k1}, P_{k2}, \dots, P_{k(d+1)}, P_{k(d+2)}\} \subset \{\alpha^k_j\}$ for each $k \in \mathcal{Q}$, our outer-approximation of the piecewise McCormick relaxation (5a)-(5c) for the constraint $W_{kk} = x_k^2$ is exact (i.e., there is no relaxation gap) at the partitioning points $x_k \in \{P_{k1}, P_{k2}, \dots, P_{k(d+1)}, P_{k(d+2)}\}$. Additionally, the piecewise McCormick relaxation (4a)-(4d) for the constraint $W_{ij} = x_i x_j$ is exact either when $x_i \in \{P_{i1}, P_{i2}, \dots, P_{i(d+1)}, P_{i(d+2)}\}$, or when $x_j \in \{P_{j1}, P_{j2}, \dots, P_{j(d+1)}, P_{j(d+2)}\}$, or both. Therefore, the point $\hat{x}$ is feasible to the original QCQP (1), which implies $\underline{v}(P) = v^*$.

Suppose instead that case (b) holds. Additionally, suppose there are multiple Y-solutions to (PMR-OA). Let $\tilde{Y} \in \mathcal{Y}$ and $\tilde{x} \in [0,1]^n$ denote the Y and x components of *another* optimal solution to problem (PMR-OA) with $\tilde{Y} \neq \hat{Y}$. Since case (a) does not hold and $\tilde{Y} \neq \hat{Y}$, we have $\tilde{x} \neq \hat{x}$. Moreover, there exist nonsingular

basis matrices $\hat{M}(P)$ and $\tilde{M}(P)$ for the LPs (PMR-OA-LP) corresponding to $\hat{Y}$ and $\tilde{Y}$, respectively, such that

$$\underline{v}(P) = v(P, \hat{Y}) = \hat{c}^{\mathrm{T}}[\hat{M}(P)]^{-1} \begin{pmatrix} d \\ \bar{B}\ \mathrm{vec}(\hat{Y}) \end{pmatrix} = \tilde{c}^{\mathrm{T}}[\tilde{M}(P)]^{-1} \begin{pmatrix} d \\ \bar{B}\ \mathrm{vec}(\tilde{Y}) \end{pmatrix} = v(P, \tilde{Y}) \tag{8}$$

for suitable vectors $\tilde{c}$ and $\hat{c}$, which include only the components of c corresponding to the basic variables of these LPs. Since not all components of $\hat{x}$ equal 0 or 1, at least some of the entries of $\hat{M}(P)$ are functions of the partitioning points P. Moreover, $v(P, \hat{Y})$ and $v(P, \tilde{Y})$ are not identical functions of P since $\tilde{x} \neq \hat{x}$. Therefore, equation (8) yields a polynomial equation in P, which implies that the set of all $P \in \mathcal{P}_d$ for which equation (8) holds has measure zero. Noting that $|\mathcal{Y}| < +\infty$ and the number of possible bases is finite for each $Y \in \mathcal{Y}$ concludes the proof. □

Algorithm 3 Strong partitioning (SP) policy for the first iteration

Input: maximum number d of partitioning points to be added per variable (excluding variable bounds).

Preprocessing steps

Initialization: partitioning vectors $\hat{P}_i^0 := (0, 1)$, $\forall i \in \mathcal{NC}$.
for $k = 1, 2, \ldots, d$ **do**
  Solve problem (PMR-OA) with the partitioning matrix $\hat{P}^{k-1}$. Let $\hat{x}^{k-1}$ denote an x-solution.
  for $i \in \mathcal{NC}$ **do**
    if $\hat{x}_i^{k-1} \approx \tilde{x}_i$ for some partitioning point $\tilde{x}_i$ in $\hat{P}_i^{k-1}$ **then**
      Set $\hat{P}_i^k = \hat{P}_i^{k-1}$.
    else
      Insert $\hat{x}_i^{k-1}$ within $\hat{P}_i^{k-1}$ to obtain a vector $\hat{P}_i^k$ satisfying $\hat{P}_{ij}^k \leq \hat{P}_{i(j+1)}^k$, $\forall j \in [\dim(\hat{P}_i^k) - 1]$.
    end if
  end for
end for
For each $i \in \mathcal{NC}$, let $n_i := \dim(\hat{P}_i^d) - 2$ and set $P_{ij}^0 := \begin{cases} 0, & \text{if } j \in [d - n_i] \\ \hat{P}_{i(j-d+n_i)}^d, & \text{if } j \in \{d + 1 - n_i, \ldots, d + 2\} \end{cases}$.
For each $i \in \mathcal{NC}$, fix variables $P_{i1}, \ldots, P_{i(d+1-n_i)}$ to 0 and variables $P_{i(d+2)}$ to 1 while solving (SP).
Preprocessing output: initial guess (and variable fixings) P^0 for problem (SP).

Solving the strong partitioning problem (SP)

Solve the max-min problem (SP) using the initial guess (and variable fixings) P^0 to obtain a solution $P^1 \in \mathcal{P}_d$ with objective $\bar{v} := \underline{v}(P^1)$.

Postprocessing steps

for $j = 2, 3, \ldots, d + 1$ **do**
  for $i \in \mathcal{NC}$ **do**
    Set $P = P^1$, fix $P_{ij} = 0$, and sort P such that $P_{ik} \leq P_{i(k+1)}$, $\forall k \in [d + 1]$.
    Solve problem (PMR-OA) with this partitioning matrix P to obtain a lower bound $\hat{v} := \underline{v}(P)$.
    if $\hat{v} \geq \bar{v} - 10^{-6}|\bar{v}|$ **then**
      Update $P^1 = P$.
    end if
  end for
end for
Postprocessing output: partitioning matrix P^1 used to construct piecewise McCormick relaxations in the first iteration of Algorithm 1. (In practice, redundant partitioning points are not added.)

4.2 Strong partitioning policy

Algorithm 3 details the strong partitioning policy for the first iteration. It includes preprocessing steps to mitigate the computational burden of solving problem (SP) to local optimality, and postprocessing steps to enable the ML model outlined in Section 5 to more effectively imitate this policy. The preprocessing heuristics determine an initial guess P^0 for the solution of the max-min problem (SP). They also eliminate a subset of the partitioning points P by fixing them to zero, which can reduce both the per-iteration cost and the number of iterations required for the bundle solver to converge. Despite these enhancements, we observe in our numerical experiments that solving the max-min problem (SP) can still be computationally prohibitive. Therefore, in the next section, we propose a practical off-the-shelf ML model to imitate this strong partitioning policy for homogeneous families of QCQPs. The postprocessing heuristics in Algorithm 3 remove (redundant) partitioning points in the solution P^1 of the max-min problem that do not significantly impact the piecewise McCormick relaxation lower bound. Reducing the number of nontrivial partitioning points can enhance the ability of ML models to effectively imitate the SP policy.

The output partitioning matrix P^1 of Algorithm 3 is used to construct piecewise McCormick relaxations in the first iteration of Algorithm 1. If the lower bound LBD obtained using P^1 and the upper bound UBD have not converged, the strong partitioning policy reverts to the heuristic partitioning policy implemented in Alpine (see Section 6.4 for details) to specify the partitioning matrices P^l for iterations $l \geq 2$ of Algorithm 1.

5 ML approximation of strong partitioning for homogeneous QCQPs

While strong partitioning can yield more effective partitioning points than existing heuristic partitioning methods, solving the max-min problem (SP) to local optimality can be time-consuming, making its direct application within Algorithm 1 impractical. To address this, we propose using AdaBoost regression models [19, 21], with regression tree base estimators [12], to imitate the strong partitioning policy for homogeneous QCQP instances. The trained AdaBoost models are then used to efficiently specify the partitioning matrix P^1 for constructing piecewise McCormick relaxations in the first iteration of Algorithm 1 for new QCQP instances from the family.

Our AdaBoost regression models sequentially train a series of regression trees, starting with an initial regression tree that fits the data. Each subsequent regression tree is trained by placing more emphasis on the training instances that the previous models struggled to predict accurately, achieved by adjusting the relative weights of those instances. The final predictions of the AdaBoost regression models are then obtained by combining the outputs of all the regression tree models using a weighted average. By iteratively focusing on the most challenging training instances, these models capture complex relationships between the input features and the output strong partitioning points while mitigating overfitting. However, a potential drawback of AdaBoost regression models is their complexity, which can make them difficult to interpret.

Algorithm 4 outlines the machine learning-based partitioning policy for the first iteration. It takes as input a feature vector for each of N QCQP instances from the family, along with the strong partitioning points determined using Algorithm 3 for these instances. The algorithm begins by splitting the training data into K disjoint folds. For each fold $k \in [K]$, it excludes the feature and strong partitioning data from that fold and learns a separate AdaBoost regression model for each partitioning point P^1_{ij}, $i \in \mathcal{NC}$ and $j \in \{2, \ldots, d+1\}$, to imitate the strong partitioning policy for specifying that partitioning point. This trained regression model is then used to predict the strong partitioning points $\{P^{1,l}_{ij}\}_{l \in \mathcal{I}_k}$ for the instances in the kth fold.

The output partitioning matrix $\hat{P}^{1,l}$ from Algorithm 4 is used to construct piecewise McCormick relaxations at the first iteration of Algorithm 1 for the lth QCQP instance. Similar to the strong partitioning policy, this AdaBoost-based policy reverts to the heuristic partitioning policy implemented in Alpine (see Section 6.4 for details) to specify the partitioning matrices from the second iteration of Algorithm 1.

We use the following instance-specific features as inputs to our AdaBoost regression models:

(i) Parameters θ that uniquely parametrize each QCQP instance in the family (see Section 6.1 for details).

Algorithm 4 ML-based partitioning policy for the first iteration

Input: maximum number d of partitioning points per variable (excluding variable bounds), data $\{(f^l, P^{1,l})\}_{l=1}^N$, where $f^l \in \mathbb{R}^{d_f}$ represents a feature vector and $P^{1,l} \in \mathcal{P}_d$ denotes the strong partitioning points determined using Algorithm 3 for the lth QCQP instance, number of data folds $K \in \{2, \ldots, N\}$, maximum number N_{wl} of weak learners for the `AdaBoostRegressor` model, and maximum depth D_{max} of its `DecisionTreeRegressor` base estimator models.

Preprocessing: randomly split the set of instance indices $[N]$ into K disjoint folds of (approximately) equal size. Let $\mathcal{I}_k \subset [N]$ denote the set of indices in the kth fold for each $k \in [K]$.

for $k = 1, 2, \ldots, K$ **do**
  for $i \in \mathcal{NC}$ **do**
    for $j = 2, 3, \ldots, d+1$ **do**
      Gather the training data $\mathcal{T}_{ijk} := \{(f^l, P_{ij}^{1,l})\}_{l \in [N] \setminus \mathcal{I}_k}$ consisting of input-output pairs corresponding to the jth partitioning point of variable x_i, omitting the kth data fold.
      Train an `AdaBoostRegressor` model on $\mathcal{T}_{ijk}$ using a `DecisionTreeRegressor` as the base estimator with maximum depth D_{max}, and a maximum of N_{wl} weak learners. (All other hyperparameters for `AdaBoostRegressor` and `DecisionTreeRegressor` are set to their default values.)
      Use the trained `AdaBoostRegressor` model along with the features $\{f^l\}_{l \in \mathcal{I}_k}$ to generate predictions $\{\hat{P}_{ij}^{1,l}\}_{l \in \mathcal{I}_k}$, with each $\hat{P}_{ij}^{1,l} \in [0, 1]$, for the strong partitioning points $\{P_{ij}^{1,l}\}_{l \in \mathcal{I}_k}$ for instances in the kth data fold.
    end for
    for $l \in \mathcal{I}_k$ **do**
      Sort the predictions $\{\hat{P}_{ij}^{1,l}\}_{j=2}^{d+1}$ such that $\hat{P}_{ij}^{1,l} \leq \hat{P}_{i(j+1)}^{1,l}$, $\forall j \in \{2, \ldots, d\}$.
    end for
  end for
end for

Output: K-fold out-of-sample ML predictions $\{\hat{P}^{1,l}\}_{l=1}^N$, where $\hat{P}^{1,l} \in \mathcal{P}_d$ is used to construct piecewise McCormick relaxations in the first iteration of Algorithm 1 for the lth QCQP instance.

(ii) The best found feasible solution during presolve, obtained via a single local solve of (QCQP).

(iii) The McCormick lower bounding solution, obtained by solving the convex problem (2).

Although it is theoretically sufficient to use only the parameters θ as features, since they uniquely identify each QCQP instance in the family, we also include features (ii) and (iii) as they are relatively inexpensive to compute and intuitively help inform the partitioning strategy. These additional features are complex transformations of the parameters θ, which might be difficult to uncover otherwise.

In contrast with much of the literature on learning for MILPs, we train separate ML models for each QCQP family since both the feature and output dimensions of our AdaBoost regression models depend on the problem dimensions. While we plan to design more advanced ML architectures that can accommodate variable feature and output dimensions in future work, we do not view the need to train different ML models for each QCQP family to be a major limitation. This is because decision-makers often care about solving instances of the *same* underlying QCQP with only a few varying model parameters, which means they only need to train a single set of ML models with fixed feature and output dimensions for their QCQP family.

Finally, we mention that we briefly explored the use of neural network models as alternatives to our AdaBoost regression models. Although we do not provide detailed results, we found that our AdaBoost regression models significantly outperformed these simple neural networks, both in predicting the strong partitioning points and in the performance of the predicted points when used to specify the partitions in the first iteration of Algorithm 1. We posit that AdaBoost models might perform better than neural networks for the following reasons. First, AdaBoost models tend to identify redundant strong partitioning points more

accurately. The preprocessing and postprocessing heuristics in Algorithm 3 are designed to set partitioning points to zero (making them redundant) if they are not expected to significantly impact the piecewise McCormick relaxation lower bound. We observe that our AdaBoost models typically predict these redundant partitioning points as zero. Consequently, using AdaBoost predictions to inform the partitioning matrix P^1 in Algorithm 1 results in easier piecewise McCormick relaxation problems (PMR) in the first iteration, leading to significant computational speedups. Next, the optimal solution of the max-min problem (SP) is often discontinuous when viewed as a function of the QCQP instance features. AdaBoost models may be better suited to predict this discontinuous mapping from the features of a QCQP instance to an optimal solution of (SP), particularly in cases where training data is scarce.

In the next section, we evaluate both the quality of our AdaBoost regression model predictions and their effectiveness when used to specify the partitioning matrix P^1 in the first iteration of Algorithm 1.

6 Numerical experiments

We describe the methodology for generating our random test instances in Section 6.1 and outline our computational setup and the metrics used in our experiments in Section 6.2. We benchmark the difficulty of our test instances using the solvers BARON and Gurobi in Section 6.3. Section 6.4 details the partitioning policy implemented within Alpine, which serves as a benchmark for our strong partitioning and AdaBoost regression-based policies. In Section 6.5, we evaluate the effectiveness of the AdaBoost models in imitating the strong partitioning policy. Finally, Section 6.6 compares the performance of Alpine's default partitioning policy with the strong partitioning and AdaBoost policies when used to select Alpine's partitioning points.

6.1 Test instances

We describe the process of generating homogeneous families of random QCQPs, motivated by real-world applications that require solving the same underlying QCQP with slightly varying model parameters. Although QCQP instances are available from libraries such as QPLIB [22], we are not aware of any libraries containing *homogeneous* QCQP instances. To address this need, we generate random homogeneous QCQP instance families by building on instance generation schemes from the literature. Python scripts for generating these QCQP families can be found at `https://github.com/lanl-ansi/Alpine.jl/tree/master/examples/random_QCQPs`.

6.1.1 Random bilinear programs

Based on the numerical study in Bao et al. [5], we consider the following family of parametric bilinear programs that are parametrized by $\theta \in [-1,1]^{d_\theta}$ to reflect real-world applications where the same underlying QCQP is solved with slightly varying parameters:

$$\begin{aligned} v(\theta) := \min_{x\in[0,1]^n} \quad & x^{\mathrm{T}} Q^0(\theta) x + (r^0(\theta))^{\mathrm{T}} x \\ \text{s.t.} \quad & x^{\mathrm{T}} Q^i(\theta) x + (r^i(\theta))^{\mathrm{T}} x \le b_i, \quad \forall i \in [m_I], \\ & (a^j)^{\mathrm{T}} x = d_j, \quad \forall j \in [m_E], \end{aligned}$$

where vectors $r^k(\theta) \in \mathbb{R}^n$, $\forall k \in \{0\} \cup [m_I]$, $a^j \in \mathbb{R}^n$, $\forall j \in [m_E]$, $b \in \mathbb{R}^{m_I}$, $d \in \mathbb{R}^{m_E}$, and the matrices $Q^k(\theta) \in \mathcal{S}^n$, $\forall k \in \{0\} \cup [m_I]$, are not assumed to be positive semidefinite.

We generate 1000 instances each with $n \in \{10, 20, 50\}$ variables and $|\mathcal{B}| = \min\{5n, \binom{n}{2}\}$ bilinear terms, $|\mathcal{Q}| = 0$ quadratic terms, $m_I = n$ bilinear inequalities, and $m_E = 0.2n$ linear equalities [5]. Each family of instances with a fixed dimension n is constructed to have the same set of $|\mathcal{B}|$ bilinear terms. We set the dimension $d_\theta = 3 \times (0.2m_I + 1)$ (an explanation for this choice is provided below). The problem data is generated as follows [cf. 5]. All entries of the vectors a^j and d are drawn i.i.d. from the uniform distribution

$U(-1,1)$, while all entries of the vector b are drawn i.i.d. from $U(0,100)$. The components of θ are drawn i.i.d. from $U(-1,1)$. Each Q^k and r^k, $k \in \{0,1,\ldots,0.2m_I\}$, is of the form:

$$Q^k(\theta) = \bar{Q}^k + \sum_{l=3k+1}^{3k+3} \theta_l \tilde{Q}^{k,l-3k}, \qquad r^k(\theta) = \bar{r}^k + \sum_{l=3k+1}^{3k+3} \theta_l \tilde{r}^{k,l-3k}.$$

The nonzero entries of $\bar{Q}^k \in \mathcal{S}^n$ are drawn i.i.d. from $U(-0.5,0.5)$, whereas the nonzero entries of $\bar{r}^k \in \mathbb{R}^n$ are drawn i.i.d. from $U(-1,1)$. The matrices $\tilde{Q}^{k,l} \in \mathcal{S}^n$, $k \in \{0\} \cup [0.2m_I]$, $l \in [3]$, are generated as follows. For each $(i,j) \in \mathcal{B}$, $k \in \{0\} \cup [0.2m_I]$, and $l \in [3]$, we set $\tilde{Q}^{k,l}_{ij} := \Gamma^{k,l}_{ij} \bar{Q}^k_{ij}$, where $\Gamma^{k,l}_{ij}$ are drawn i.i.d. from $U(0,0.5)$. We set $\tilde{Q}^{k,l}_{ij} = \tilde{Q}^{k,l}_{ji}$ for each $(i,j) \in \mathcal{B}$ to ensure that $\tilde{Q}^{k,l}$ is symmetric. Next, for each $i \in [n]$, $k \in \{0\} \cup [0.2m_I]$, and $l \in [3]$, we set $\tilde{r}^{k,l}_i := \delta^{k,l}_i \bar{r}^k_i$, where $\delta^{k,l}_i$ are drawn i.i.d. from $U(0,0.5)$. The data $\bar{Q}^k$, $\tilde{Q}^{k,l}$, $\bar{r}^k$, and $\tilde{r}^{k,l}$ are fixed across the 1000 instances for each value of n. Since each $\tilde{Q}^{k,l}$ and $\tilde{r}^{k,l}$ is a different perturbation of $\bar{Q}^k$ and $\bar{r}^k$, the expansions of Q^k and r^k can be motivated using principal component analysis. The nonzero entries of Q^k and r^k, $k \in \{0.2m_I+1,\ldots,m_I\}$, are identical across all 1000 instances for each value of n, with each entry of Q^k drawn i.i.d. from $U(-0.5,0.5)$ and each entry of r^k drawn i.i.d. from $U(-1,1)$. Finally, the constraint coefficients are re-scaled so that all entries of the vectors b and d equal one. Note that for a fixed dimension n, each instance is uniquely specified by the parameters θ of dimension $d_\theta = 3(0.2n+1)$. Given that we assume the forms of each $Q^k(\theta)$ and $r^k(\theta)$, $k \in \{0\} \cup [m_I]$, are known, with only the parameter θ varying across different instances within each QCQP family, we can use θ as part of the features for each QCQP instance.

6.1.2 Random QCQPs with bilinear *and* univariate quadratic terms

We also generate 1000 random QCQPs with $|\mathcal{B}| = \min\{5n, \binom{n}{2}\}$ bilinear terms and $|\mathcal{Q}| = \lfloor 0.25n \rfloor$ univariate quadratic terms for each of $n \in \{10,20,50\}$ variables. Once again, all instances for a fixed dimension n comprise the same set of bilinear and univariate quadratic terms. The coefficients of quadratic terms in the objective and constraints are generated similarly to the coefficients of bilinear terms in Section 6.1.1. The rest of the model parameters and problem data (including θ) are also generated similarly as in Section 6.1.1.

6.1.3 The pooling problem

The pooling problem is a classic example of a bilinear program introduced by Haverly [26]. It has several important applications, including petroleum refining [30, 61], natural gas production [30, 37], and wastewater treatment [7, 44, 55]. Its goal is to blend inputs of differing qualities at intermediate pools to produce outputs that meet quality specifications while satisfying capacity constraints at inputs, pools, and outputs. Solving the pooling problem is in general NP-hard.

We consider instances of the pooling problem with 45 inputs, 15 pools, 30 outputs, and a single quality. Each instance has 116 input-output arcs, 71 input-pool arcs, and 53 pool-output arcs, yielding 572 variables and 621 constraints, including 360 linear constraints and 261 bilinear equations (with 124 variables involved in bilinear terms). We use the *pq*-formulation of the pooling problem from Section 2 of Luedtke et al. [41]. Unlike the random bilinear instances in Section 6.1.1 where all of the original "x variables" participate in bilinear terms, only 124 out of the 311 original variables in the pooling model participate in bilinear terms.

We first generate a nominal instance using the "random Haverly" instance generation approach (see https://github.com/poolinginstances/poolinginstances for details) in [41] that puts together 15 perturbed copies of one of the Haverly [26] pooling instances and adds 150 edges to it. We modify the target output quality concentrations generated by Luedtke et al. [41] to construct harder instances. For each output j, we compute the minimum $c^{\min}_j$ and maximum $c^{\max}_j$ input concentrations of the quality over the subset of inputs from which there exists a path to output j. We then specify the lower and upper bound on the quality concentration at output j to be $c^{\min}_j + \alpha_j(c^{\max}_j - c^{\min}_j)$ and $c^{\min}_j + \beta_j(c^{\max}_j - c^{\min}_j)$, respectively, where $\alpha_j \sim U(0.2,0.4)$ and $\beta_j \sim U(0.6,0.8)$ are generated independently. We also rescale the capacities of the inputs, pools, and outputs and the costs of the arcs for better numerical performance. Note that while

all variables in the formulation are nonnegative, upper bounds on the variables are not necessarily equal to one after rescaling. After constructing a nominal instance using the above procedure, we use it to generate 1000 random pooling instances by randomly perturbing each input's quality concentration (parameters θ for this problem family) by up to 20%, uniformly and independently.

6.2 Computational setup and evaluation metrics for our numerical experiments

6.2.1 Computational setup

We benchmark the difficulty of our test instances using BARON and Gurobi. We use Gurobi 9.1.2 via Gurobi.jl v0.11.3 and BARON 23.6.23 via BARON.jl v0.8.2, with the option of using CPLEX 22.1.0 as BARON's MILP solver. (Note that BARON does not support Gurobi as an MILP solver.) Each BARON and Gurobi run is assigned a time limit of 2 hours, with target relative and absolute optimality gaps of 10^{-4} and 10^{-9}. All other settings in BARON and Gurobi were kept at default.

We evaluate the performance of different partitioning policies within Alpine (`https://github.com/lanl-ansi/Alpine.jl`). We use Julia 1.6.3 and JuMP.jl v1.1.1 to formulate our test instances and Alpine.jl v0.4.1 to solve them. We use Gurobi 9.1.2 via Gurobi.jl v0.11.3 with `MIPGap` $= 10^{-6}$ for solving LPs, MILPs, and convex mixed-integer QCQPs within Alpine. We use Ipopt 3.14.4 via Ipopt.jl v1.0.3 (with `max_iter` $= 10^4$) to solve the random bilinear and QCQP instances in Sections 6.1.1 and 6.1.2 locally within Alpine. To solve the random pooling instances in Section 6.1.3 locally within Alpine, we switch to Artelys Knitro 12.4.0 via KNITRO.jl v0.13.0 (with `algorithm` $= 3$) due to Ipopt's reduced effectiveness for these instances. Each Alpine run is assigned a time limit of 2 hours, with target relative and absolute optimality gaps of 10^{-4} and 10^{-9}. (Alpine's definition of relative gap differs *slightly* from those of BARON and Gurobi, as noted in Section 6.2.2.) We deactivate bound tightening techniques within Alpine due to their ineffectiveness on our medium and large-scale instances. We partition the domains of all variables involved in nonconvex terms within Alpine and set its remaining options to their default values.

We evaluate the performance of the strong partitioning and ML-based policies when used to specify the partitioning matrix P^1 in Alpine's first iteration. Algorithm 3 for strong partitioning is implemented in Julia 1.6.3 and integrated within Alpine.jl v0.4.1 (`https://github.com/lanl-ansi/Alpine.jl`). To solve problem (SP) to local optimality, we use the bundle solver MPBNGC 2.0 [42] via MPBNGCInterface.jl (`https://github.com/milzj/MPBNGCInterface.jl`), with the options `OPT_LMAX` $= 20$, `OPT_EPS` $= 10^{-9}$, and `OPT_NITER` $=$ `OPT_NFASG` $= 500$. We do not specify a time limit for Algorithm 3. We consider strong partitioning with either two or four partitioning points per partitioned variable ($d = 2$ or $d = 4$) in addition to the variable bounds. We use scikit-learn v0.23.2 [53] to design our AdaBoost regression-based approximation of strong partitioning, setting $N = 1000$, $K = 10$, $N_{wl} = 1000$, $D_{max} = 25$, and either $d = 2$ or $d = 4$ in Algorithm 4. We do not carefully tune the hyperparameters in Algorithm 4, as our primary focus is on the performance of partitioning points prescribed by our AdaBoost-based policy when used within Algorithm 1.

All of our experiments were conducted on nodes of the Darwin cluster at LANL, which are equipped with dual socket Intel Broadwell 18-core processors (E5-2695_v4 CPUs, base clock rate of 2.1 GHz), EDR InfiniBand, and 125 GB of memory. Each instance was run exclusively on a single node, and different solution approaches were executed sequentially to minimize the impact of variability in machine performance.

6.2.2 Evaluation metrics for our numerical experiments

We benchmark the difficulty of our test instances using the global solvers BARON and Gurobi with the following metrics:

i. **Solution times:** shifted geometric mean (GM), median, minimum, and maximum solution times.

ii. **Number of iterations or B&B nodes:** GM and median number of iterations or B&B nodes explored.

iii. **Number of instances solved:** the number of instances solved within the time limit and the GM of the residual optimality gap for the unsolved instances.

The shifted geometric mean of a positive vector t of solution times (in seconds) is defined as:

$$\text{Shifted GM}(t) = \exp\Big(\frac{1}{N}\sum_{i=1}^{N}\ln(t_i + 10)\Big) - 10.$$

The residual optimality gap for an unsolved instance is defined as:

$$\text{TLE Gap } = \frac{\text{UB} - \text{LB}}{10^{-6} + |\text{UB}|},$$

where UB and LB are the upper and lower bounds on the optimal value returned by the solver at termination.

We assess the effectiveness of the AdaBoost regression models in imitating the strong partitioning policy by evaluating the scaled mean absolute errors (MAEs) of their out-of-sample predictions for the strong partitioning points. We then compare the performance of Alpine's default partitioning policy with the use of the strong partitioning and AdaBoost-based policies within Alpine using the following metrics:

i. **Solution times:** shifted GM, median, minimum, and maximum solution times, as well as statistics on the speedup or slowdown relative to the performance of Alpine's default partitioning policy.

ii. **Effective optimality gaps:** GM, median, minimum, and maximum effective optimality gap after Alpine's first iteration, along with the percentage of instances for which the effective optimality gap reaches the minimum value of 10^{-4} after the first iteration.

iii. **Number of instances solved:** the number of instances solved within the time limit and the GM of the residual optimality gap for the unsolved instances.

We also plot solution profiles comparing the performance of different partitioning policies within Alpine, and histograms showing the reduction in effective optimality gaps achieved by using the strong partitioning and ML-based policies relative to Alpine's default partitioning policy. We do not include performance profiles due to their known issues (see http://plato.asu.edu/bench.html).

We define the *effective* relative optimality gap as:

$$\text{Effective Optimality Gap } = \max\Big\{10^{-4}, \frac{v^* - v^{\text{LBD}}}{10^{-6} + |v^*|}\Big\}, \tag{9}$$

where v^* is the optimal objective value, v^{LBD} is Alpine's lower bound after one iteration, and 10^{-4} is the target relative optimality gap. By measuring the gap of v^{LBD} relative to the optimal value v^* instead of the best found feasible solution, we do not let the performance of the local solver impact our evaluation of the different partitioning policies. Thresholding this optimality gap at 10^{-4} also lends equal importance to all optimality gaps less than this target since all such gaps are sufficient for Alpine's convergence.

6.3 Benchmarking test instances using state-of-the-art global optimization solvers

To highlight the nontrivial nature of our nonconvex QCQP test instances, we solve them to global optimality using the state-of-the-art global optimization solvers BARON and Gurobi, evaluating them with the metrics detailed in Section 6.2.2. It is important to note that our goal is not to compare BARON or Gurobi with the different versions of Alpine, but rather to demonstrate that both our QCQP instances and the accelerations of Alpine achieved are nontrivial.

6.3.1 Benchmarking using BARON

Table 1 present statistics of BARON run times across the different QCQP families. BARON solves the 10-variable and 20-variable random bilinear and QCQP instances within seconds. However, it takes over 5 minutes on average to solve the 50-variable random bilinear instances and over 20 minutes on average to

Table 1: Statistics on BARON solution times, including the shifted geometric mean, median, minimum, and maximum times over the subset of 1000 instances for which BARON does not hit the 2 hour time limit. The sixth and seventh columns show the geometric mean and median of the total number of BARON iterations across the 1000 instances. The last two columns denote the number of instances for which BARON hits the time limit and the corresponding geometric mean of residual optimality gaps at termination, respectively.

Problem Family	BARON Solution Time (seconds)				BARON Iterations			
	Shifted GM	Median	Min	Max	GM	Median	# TLE	TLE Gap (GM)
Bilinear, $n = 10$	0.2	0.2	0.1	0.4	1	1	0	—
Bilinear, $n = 20$	3.8	3.9	1.3	7.6	9	9	0	—
Bilinear, $n = 50$	312.4	281.4	57.7	5194.9	9934	9193	0	—
QCQP, $n = 10$	0.4	0.4	0.1	0.7	1	1	0	—
QCQP, $n = 20$	5.0	4.8	1.5	9.8	17	13	0	—
QCQP, $n = 50$	1311.3	1741.6	16.3	7165.4	54691	73221	107	3.2×10^{-2}
Pooling	553.5	647.2	17.0	7189.2	50954	263547	417	2.4×10^{-2}

Table 2: Statistics on Gurobi solution times, including the shifted geometric mean, median, minimum, and maximum times over the subset of 1000 instances for which Gurobi does not hit the 2 hour time limit. The sixth and seventh columns show the geometric mean and median of the total number of nodes explored by Gurobi across the 1000 instances. The last two columns denote the number of instances for which Gurobi hits the time limit and the corresponding geometric mean of residual optimality gaps at termination, respectively.

Problem Family	Gurobi Solution Time (seconds)				Gurobi B&B Nodes			
	Shifted GM	Median	Min	Max	GM	Median	# TLE	TLE Gap (GM)
Bilinear, $n = 10$	0.03	0.03	0.01	0.13	1	1	0	—
Bilinear, $n = 20$	0.7	0.7	0.3	1.3	2732	2687	0	—
Bilinear, $n = 50$	7.2	7.1	2.6	18.3	17181	17263	0	—
QCQP, $n = 10$	0.02	0.01	0.01	0.08	1	1	0	—
QCQP, $n = 20$	0.6	0.6	0.3	1.1	2235	2389	0	—
QCQP, $n = 50$	8.0	8.1	2.1	25.1	26108	27711	0	—
Pooling	63.6	44.3	1.1	6367.1	833312	739876	10	3×10^{-4}

solve the 50-variable random QCQP instances. BARON also times out on 107 out of 1000 of the 50-variable instances with univariate quadratic terms. BARON finds the random pooling instances to be significantly harder, timing out on 417 out of 1000 instances and taking roughly 9 minutes on average to solve the remaining 583 out of 1000 instances (while BARON finds global solutions, it is unable to prove global optimality within the time limit). The last column in Table 1 indicates the GM of the relative optimality gap at termination for instances where BARON hits the time limit. Finally, the sixth and seventh columns note the GM and median of the total number of BARON iterations. BARON requires significantly more iterations for the 50-variable random bilinear and QCQP instances, as well as for the pooling instances, compared to the 10-variable and 20-variable instances.

6.3.2 Benchmarking using Gurobi

Table 2 present statistics of Gurobi run times across the different QCQP families. Gurobi solves the 10-variable and 20-variable random bilinear and QCQP instances within a second and requires about 7 seconds on average to solve the 50-variable random bilinear and QCQP instances. However, Gurobi finds the random pooling instances to be relatively more challenging, timing out on 10 out of 1000 instances and taking about one minute on average to solve the remaining 990 out of 1000 instances. The last column in Table 2 notes

Algorithm 5 Alpine's adaptive partition refinement policy for iteration l

1: **Input:** partitioning matrix $P^{l-1} \in [0,1]^{|\mathcal{NC}|\times 2l}$ used to construct piecewise McCormick relaxations at iteration $l-1$, index $\mathcal{A}(i,l-1) \in [2l-1]$ of an active partition for variable x_i at iteration $l-1$, $\forall i \in \mathcal{NC}$, a reference point $\bar{x}^{l-1}$ within the active partition, i.e., with $\bar{x}_i^{l-1} \in [P^{l-1}_{i(\mathcal{A}(i,l-1))}, P^{l-1}_{i(\mathcal{A}(i,l-1)+1)}]$, $\forall i \in \mathcal{NC}$, and parameter $\Delta \geq 4$.

2: **Initialization:** set $P^l = P^{l-1}$.

3: **for** $i \in \mathcal{NC}$ **do**

4: Let $\text{width}(\mathcal{A}(i,l-1)) := P^{l-1}_{i(\mathcal{A}(i,l-1)+1)} - P^{l-1}_{i(\mathcal{A}(i,l-1))}$. Add two partitioning points $\underline{p}_i^l$ and $\bar{p}_i^l$ to the active partition $[P^{l-1}_{i(\mathcal{A}(i,l-1))}, P^{l-1}_{i(\mathcal{A}(i,l-1)+1)}]$ for variable x_i as follows:

$$P_i^l := \left(P^{l-1}_{i1}, \ldots, P^{l-1}_{i(\mathcal{A}(i,l-1))}, \underline{p}_i^l, \bar{p}_i^l, \ldots, P^{l-1}_{i(2l)}\right), \quad \text{where}$$

$$\underline{p}_i^l := \max\left\{P^{l-1}_{i(\mathcal{A}(i,l-1))}, \bar{x}_i^{l-1} - \frac{\text{width}(\mathcal{A}(i,l-1))}{\Delta}\right\},$$

$$\bar{p}_i^l := \min\left\{P^{l-1}_{i(\mathcal{A}(i,l-1)+1)}, \bar{x}_i^{l-1} + \frac{\text{width}(\mathcal{A}(i,l-1))}{\Delta}\right\}.$$

(In practice, the partitioning points $\underline{p}_i^l$ and $\bar{p}_i^l$ are not added to P_i^l if $\bar{x}_i^{l-1} - \frac{\text{width}(\mathcal{A}(i,l-1))}{\Delta} \leq P^{l-1}_{i(\mathcal{A}(i,l-1))}$ or $\bar{x}_i^{l-1} + \frac{\text{width}(\mathcal{A}(i,l-1))}{\Delta} \geq P^{l-1}_{i(\mathcal{A}(i,l-1)+1)}$, respectively.)

5: **end for**

6: **Output:** partitioning matrix P^l used to construct piecewise McCormick relaxations at iteration l.

the GM of the relative optimality gap at termination for instances where Gurobi hits the time limit. The sixth and seventh columns note the GM and median of the total number of B&B nodes explored by Gurobi. Despite requiring a significant number of nodes for convergence, particularly for the pooling instances, Gurobi solves our test instances much faster than BARON, potentially due to the use of cheaper relaxations, efficient parallelization and effective engineering.

6.4 Alpine's partitioning policy

Algorithm 5 outlines the partitioning policy implemented within Alpine [47]. It adds up to two partitioning points to the active partition for each variable x_i, $i \in \mathcal{NC}$, around a reference point $\bar{x}_i^{l-1}$. For the first iteration, $P_i^0 := (0,1)$, $\forall i \in \mathcal{NC}$, and the reference point $\bar{x}^0$ is set either to a feasible local solution from presolve, if one is found, or to a solution to the termwise McCormick relaxation (2) otherwise. In subsequent iterations ($l > 1$), $\bar{x}^{l-1}$ is specified as the x-component of a solution to the piecewise McCormick relaxation problem (PMR) at iteration $l-1$, constructed using the partitioning matrix P^{l-1}. The parameter Δ (default value $= 10$) is a dimensionless scaling factor for the size of the partition constructed around the reference point $\bar{x}^{l-1}$. A larger value of Δ results in a narrower partition around $\bar{x}^{l-1}$.

This policy empirically performs well on Alpine's test library and is motivated by the observation that uniformly partitioning variable domains [as proposed in 15, 55, 60] often creates many partitions that do not significantly improve the piecewise McCormick relaxation lower bound. However, it relies on heuristic choices that could be improved. For example, it uses the *same* parameter Δ to partition the domains of *all* variables and only considers symmetric partition refinements around the point $\bar{x}^{l-1}$. Additionally, the quality of its partitioning points P^1 in the first iteration depends on the quality of the feasible solution found during presolve, with suboptimal presolve solutions potentially leading to slow convergence. Numerical experiments in Section 6.6 demonstrate how strong partitioning and its ML approximation effectively address these limitations of Alpine's partitioning policy.

Table 3: Statistics on the scaled mean absolute errors (MAEs) of the out-of-sample predictions of the AdaBoost regression models when they are trained to predict $d = 2$ strong partitioning points per variable.

Scaled MAE	< 0.01	< 0.02	< 0.05	< 0.1	< 0.2
	% of the $2\|\mathcal{NC}\|$ Partitioning Points				
Bilinear, $n = 10$	60	75	80	95	100
Bilinear, $n = 20$	15	22.5	60	87.5	97.5
Bilinear, $n = 50$	31	39	70	94	100
QCQP, $n = 10$	65	80	95	100	100
QCQP, $n = 20$	35	37.5	77.5	92.5	100
QCQP, $n = 50$	56	66	85	99	100
Pooling	65.7	70.9	78.6	89.5	97.2

6.5 Effectiveness of the ML model in imitating the strong partitioning policy

We evaluate the effectiveness of our AdaBoost regression models in imitating the strong partitioning policy for the different QCQP families. We do not extensively tune the hyperparameters of our AdaBoost models, as our primary focus is on the performance of the partitioning points they prescribe when used within Algorithm 1. Section 6.2.1 details the parameter settings used in Algorithm 4.

Table 3 presents statistics on the scaled mean absolute errors (MAEs) of the out-of-sample predictions for the $2|\mathcal{NC}|$ partitioning points generated by the AdaBoost models. These models are trained to predict $d = 2$ strong partitioning points per variable. The MAEs are averaged over 1000 instances in each family and are scaled by the upper bounds of the corresponding x variables (which are equal to one for the random bilinear and QCQP instances). Approximately 90% or more of the partitioning points predicted by the AdaBoost models have a scaled MAE of less than 10% for each problem family, indicating that this ML framework effectively and efficiently imitates strong partitioning across different problem families.

6.6 Evaluating the performance of strong partitioning and ML-based policies in Alpine

We compare the performance of Alpine's default partitioning policy with the strong partitioning (Alpine+SP2) and AdaBoost-based (Alpine+ML2) policies, which select two partitioning points per variable in each iteration of Alpine, using the metrics detailed in Section 6.2.2. For the bilinear $n = 20$ and QCQP $n = 20$ families, we also compare these policies with the strong partitioning (Alpine+SP4) and AdaBoost-based (Alpine+ML4) policies that select four partitioning points per variable in Alpine's first iteration. We reiterate that the strong partitioning and AdaBoost policies differ from Alpine's default partitioning policy only during the first iteration of Algorithm 1. All reported times for Alpine with the strong partitioning and AdaBoost policies exclude the time required to run Algorithm 3 and Algorithm 4.

Table 4 presents statistics on the run times of Alpine with the default, strong partitioning, and AdaBoost-based partitioning policies across the different QCQP families. Table 5 records the speedup or slowdown of Alpine with the strong partitioning and AdaBoost-based policies relative to default Alpine. Table 6 provides statistics on the effective optimality gaps (9) of Alpine with the different partitioning policies after the first iteration. Table 7 reports statistics on the time required to run Algorithm 3 to determine strong partitioning points for the different QCQP families. Figures 2, 3, and 4 plot solution profiles and histograms showing the reduction in effective optimality gaps achieved after the first iteration by the strong partitioning and AdaBoost policies relative to Alpine's default partitioning policy.

Bilinear instances. Table 4 indicates that Alpine+SP2 reduces the shifted GM of default Alpine's solution time by factors of 4.5, 5.1, and 7.7 for the $n = 10$, $n = 20$, and $n = 50$ families. Alpine+ML2 offers a moderate approximation of Alpine+SP2, reducing the shifted GM of default Alpine's solution time by factors of 3.5,

Table 4: Statistics on solution times. Columns correspond to the shifted geometric mean, median, minimum, and maximum times over the subset of 1000 instances that did not hit the 2 hour time limit. The times for Alpine+SP2 and Alpine+SP4 do not include the time for running Algorithm 3 to determine strong partitioning points. The last two columns denote the number of instances for which each method hits the time limit and the corresponding geometric mean of residual optimality gaps at termination, respectively.

Problem Family	Solution Method	Solution Time (seconds)					
		Shifted GM	Median	Min	Max	# TLE	TLE Gap (GM)
Bilinear, $n = 10$	Alpine (default)	0.51	0.47	0.14	2.41	0	—
	Alpine+SP2	0.11	0.10	0.06	0.28	0	—
	Alpine+ML2	0.15	0.10	0.06	1.64	0	—
Bilinear, $n = 20$	Alpine (default)	21.4	21.9	5.1	161.5	0	—
	Alpine+SP2	4.2	2.0	0.8	132.6	0	—
	Alpine+ML2	10.0	7.8	1.1	116.0	0	—
	Alpine+SP4	2.4	1.9	0.8	94.2	0	—
	Alpine+ML4	9.3	7.2	1.0	117.4	0	—
Bilinear, $n = 50$	Alpine (default)	405.9	336.2	48.0	7135.9	24	4.4×10^{-4}
	Alpine+SP2	52.8	34.9	4.2	5705.1	4	1.6×10^{-4}
	Alpine+ML2	101.6	83.6	6.6	7071.7	5	1.8×10^{-4}
QCQP, $n = 10$	Alpine (default)	0.85	0.81	0.62	2.29	0	—
	Alpine+SP2	0.10	0.09	0.07	0.27	0	—
	Alpine+ML2	0.27	0.12	0.07	2.89	0	—
QCQP, $n = 20$	Alpine (default)	40.1	35.6	4.6	241.1	0	—
	Alpine+SP2	7.7	1.7	0.8	135.4	0	—
	Alpine+ML2	13.0	9.5	1.0	180.1	0	—
	Alpine+SP4	2.4	1.5	0.7	125.7	0	—
	Alpine+ML4	9.4	6.4	0.9	101.2	0	—
QCQP, $n = 50$	Alpine (default)	391.5	289.1	36.6	7198.2	0	—
	Alpine+SP2	63.3	51.9	4.2	6055.2	0	—
	Alpine+ML2	100.5	118.2	5.3	6514.2	0	—
Pooling	Alpine (default)	242.8	212.5	25.9	7091.9	7	2.9×10^{-4}
	Alpine+SP2	66.7	49.7	1.6	6127.1	5	2.1×10^{-4}
	Alpine+ML2	117.1	101.9	11.4	6097.0	1	2.8×10^{-4}

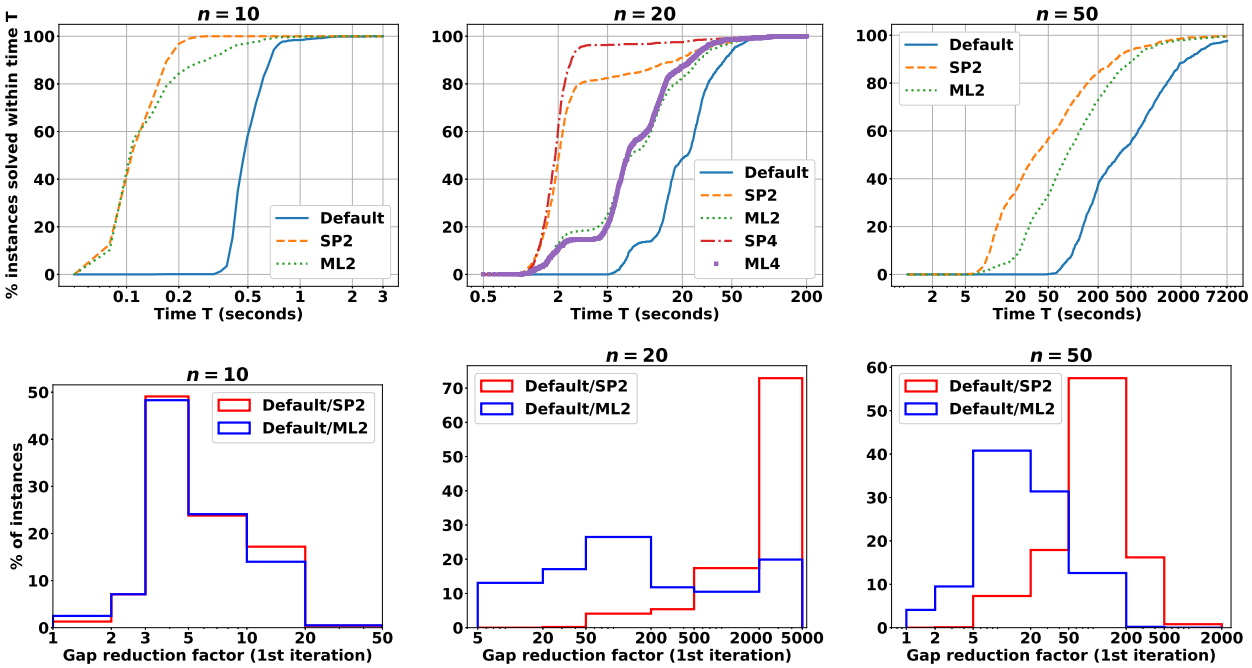
2.1, and 4, respectively, for $n = 10$, $n = 20$, and $n = 50$. For the $n = 20$ instances, Alpine+SP4 and Alpine+ML4 reduce the shifted GM of default Alpine's solution time by factors of 9 and 2.3.

Table 5 shows that Alpine+SP2 achieves at least 5× speedup over default Alpine on 41.3% of the $n = 10$ instances, and at least 10× speedup on 39.9% and 46.1% of the $n = 20$ and $n = 50$ instances. Alpine+ML2 achieves at least 5× speedup on 40.1%, 22.2%, and 45.2% of the $n = 10$, $n = 20$, and $n = 50$ instances. Alpine+SP2 achieves a maximum speedup of 15×, 49×, and 685× on the $n = 10$, $n = 20$, and $n = 50$ instances, while Alpine+ML2 achieves a maximum speedup of 13×, 38×, and 197× on these instances.

It is important to note that the times for Alpine+SP2 and Alpine+SP4 do not include the time required to run Algorithm 3 to determine strong partitioning points, making these solution times unachievable in practice. Instead, these times reflect the potential performance that could be realized by an efficient ML model perfectly imitating the strong partitioning strategy. These results underscore the significant potential of strong partitioning as an expert strategy for accelerating the global optimization of nonconvex QCQPs.

Table 6 shows that Alpine+SP2 reduces the GM of default Alpine's effective optimality gap (9) after the

Figure 2: Results for the random bilinear instances. The times for Alpine+SP2 and Alpine+SP4 do not include the time for running Algorithm 3 to determine strong partitioning points. Top row: solution profiles indicating the percentage of instances solved by the different methods within time T seconds (higher is better). Bottom row: histograms of the ratios of the effective optimality gaps (9) of default Alpine with Alpine+SP2 and with Alpine+ML2 after one iteration (larger is better).



first iteration by factors of 5.5, 2200, and 80, respectively, for the $n = 10$, $n = 20$, and $n = 50$ instances. Alpine+ML2 also reduces the GM of default Alpine's gap after the first iteration by factors of 4.6, 180, and 15 for $n = 10$, $n = 20$, and $n = 50$. Notably, Alpine+SP2 closes the gap in the first iteration for 100%, 82.3%, and 46% of the $n = 10$, $n = 20$, and $n = 50$ instances, while default Alpine is able to close the gap in the first iteration for at most 0.1% of the instances across these families, highlighting the effectiveness of strong partitioning. Table 4 also shows that Alpine+SP2 and Alpine+ML2 terminate with smaller average gaps on the $n = 50$ instances where they time out, compared to default Alpine.

QCQP instances. Table 4 shows that Alpine+SP2 reduces the shifted GM of default Alpine's solution time by factors of 8.4, 5.2, and 6.2 for the $n = 10$, $n = 20$, and $n = 50$ families. Alpine+ML2 provides a moderate approximation, reducing the shifted GM of default Alpine's solution time by factors of 3.1, 3.1, and 3.9 for these instances. For the $n = 20$ instances, Alpine+SP4 and Alpine+ML4 reduce the shifted GM of default Alpine's solution time by factors of 16.4 and 4.3, respectively.

Table 5 indicates that Alpine+SP2 achieves at least a 10× speedup over default Alpine on 20.5%, 54.6%, and 42.1% of the $n = 10$, $n = 20$, and $n = 50$ instances. Alpine+ML2 provides at least a 5× speedup on 65.7%, 34.7%, and 42.7% of these instances. Alpine+SP2 achieves a maximum speedup of 22×, 87×, and 98× on the $n = 10$, $n = 20$, and $n = 50$ instances, while Alpine+ML2 results in a maximum speedup of 19×, 56×, and 32× on these instances. We reiterate that the times for Alpine+SP2 and Alpine+SP4 do not include the time required for running Algorithm 3. These times reflect the potential performance that could be realized by an efficient ML model perfectly imitating the strong partitioning strategy.

Table 6 shows that Alpine+SP2 reduces the GM of default Alpine's effective gap (9) after the first iteration by factors of 13, 300, and 50 for the $n = 10$, $n = 20$, and $n = 50$ instances. Alpine+ML2 reduces

Figure 3: Results for the random QCQP instances. The times for Alpine+SP2 and Alpine+SP4 do not include the time for running Algorithm 3 to determine strong partitioning points. Top row: solution profiles indicating the percentage of instances solved by the different methods within time T seconds (higher is better). Bottom row: histograms of the ratios of the effective optimality gaps (9) of default Alpine with Alpine+SP2 and with Alpine+ML2 after one iteration (larger is better).

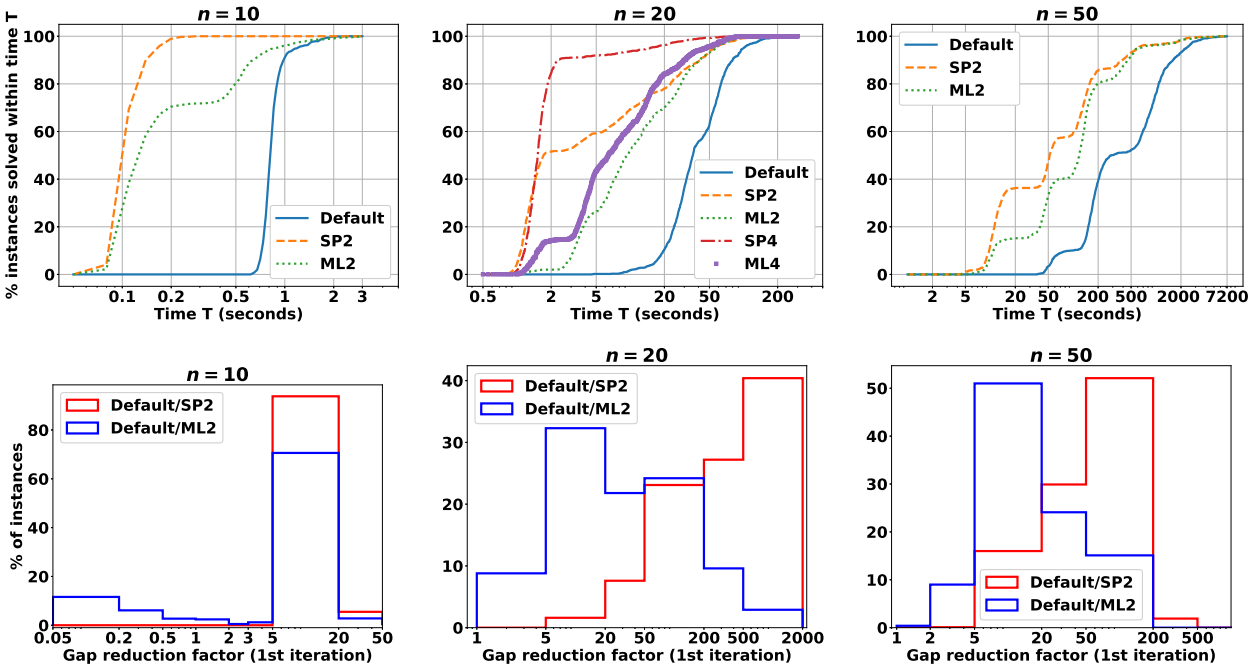
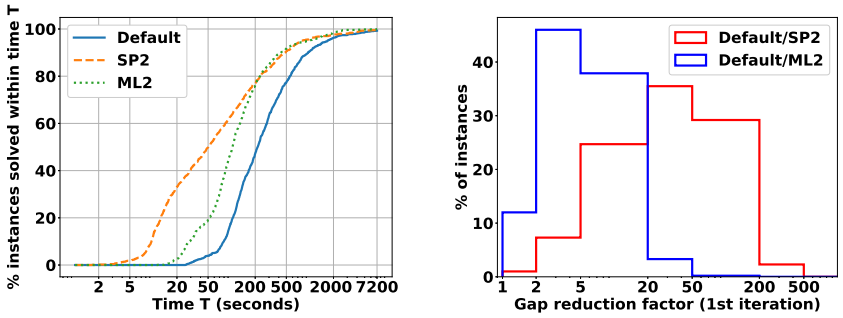


Figure 4: Results for the random pooling instances. The times for Alpine+SP2 do not include the time for running Algorithm 3 to determine strong partitioning points. Left plot: solution profile indicating the percentage of instances solved by the different methods within time T seconds (higher is better). Right plot: histograms of the ratios of the effective optimality gaps (9) of default Alpine with Alpine+SP2 and with Alpine+ML2 after one iteration (larger is better).



the GM of default Alpine's gap by factors of 4.3, 31, and 17 for these instances. Notably, Alpine+SP2 closes the gap in the first iteration for 100%, 52.2%, and 39% of the $n = 10$, $n = 20$, and $n = 50$ instances, while default Alpine is unable to close the gap in the first iteration for any of these instances.

Table 5: Statistics on the speedup or slowdown of Alpine with the strong partitioning and ML-based policies relative to default Alpine. The speedups for Alpine+SP2 and Alpine+SP4 do not account for the time spent in running Algorithm 3 to determine strong partitioning points.

Problem Family	Solution Method	Speedup/Slowdown Factor							
		< 0.5	$0.5 - 1$	$1 - 2$	$2 - 5$	$5 - 10$	$10 - 20$	$20 - 50$	> 50
Bilinear, $n = 10$	% Alpine+SP2 inst.	—	—	1.1	57.6	40.1	1.2	0	0
	% Alpine+ML2 inst.	0.2	2.1	7.7	49.9	40.0	0.1	0	0
Bilinear, $n = 20$	% Alpine+SP2 inst.	0.2	3.3	7.2	18.2	31.2	29.9	10.0	0.0
	% Alpine+ML2 inst.	3.3	9.8	25.5	39.2	15.3	6.0	0.9	0.0
	% Alpine+SP4 inst.	0.2	0.7	1.3	13.4	32.7	37.1	14.5	0.1
	% Alpine+ML4 inst.	2.8	10.5	23.3	41.4	15.2	5.9	0.9	0.0
Bilinear, $n = 50$	% Alpine+SP2 inst.	0.4	1.3	7.2	18.7	26.3	24.3	14.9	6.9
	% Alpine+ML2 inst.	0.7	4.7	16.9	32.5	25.3	13.7	5.4	0.8
QCQP, $n = 10$	% Alpine+SP2 inst.	—	—	0.1	3.3	76.1	20.4	0.1	0
	% Alpine+ML2 inst.	1.0	3.9	20.9	8.5	53.4	12.3	0	0
QCQP, $n = 20$	% Alpine+SP2 inst.	0.1	3.2	12.2	18.4	11.5	19.4	32.6	2.6
	% Alpine+ML2 inst.	0.5	5.1	19.0	40.7	23.1	9.6	1.9	0.1
	% Alpine+SP4 inst.	0	0.2	1.3	3.8	5.5	28.2	53.7	7.3
	% Alpine+ML4 inst.	0	2.9	11.6	33.3	27.0	17.2	7.6	0.4
QCQP, $n = 50$	% Alpine+SP2 inst.	0.9	1.3	10.7	22.0	23.0	32.5	7.2	2.4
	% Alpine+ML2 inst.	1.4	4.0	19.5	32.4	22.7	16.6	3.4	0
Pooling	% Alpine+SP2 inst.	2.2	6.4	19.7	26.0	21.8	16.8	6.7	0.4
	% Alpine+ML2 inst.	2.1	11.5	34.5	40.4	9.8	1.4	0.3	0

Pooling instances. Table 4 indicates that Alpine+SP2 and Alpine+ML2 reduce the shifted GM of default Alpine's solution time by factors of 3.6 and 2.1. Table 5 shows that Alpine+SP2 and Alpine+ML2 achieve at least a 5× speedup over default Alpine on 45.7% and 11.5% of the instances. Table 6 reveals that Alpine+SP2 and Alpine+ML2 reduce the GM of default Alpine's effective optimality gap (9) after the first iteration by factors of 28 and 4.5. After the first iteration, Alpine+SP2 closes the effective optimality gap for 45.2% of the instances, while default Alpine fails to close the gap for any of these instances. Finally, Alpine+SP2 and Alpine+ML2 result in maximum speedups of 120× and 41×. We reiterate that the times for Alpine+SP2 and Alpine+SP4 do not include the time required for running Algorithm 3.

Summary. Tables 4 to 6 and Figures 2 to 4 clearly demonstrate the advantages of our SP and AdaBoost policies over Alpine's default partitioning policy. Comparing Alpine+SP4 and Alpine+SP2 reveals that using SP to select more partitioning points results in more effective, non-myopic partition refinement, albeit at a significant increase in the time required to solve the max-min problem (SP). Table 7 indicates that solving the max-min problem (SP) to local optimality can be time-consuming, making the direct use of the SP policy in Alpine impractical. However, the performance metrics for Alpine+SP2 and Alpine+SP4 suggest significant potential improvements, reflecting the best-case scenario achievable by an efficient ML model that perfectly imitates the SP policy. This highlights the substantial potential for accelerating the global optimization of QCQPs through tailored ML approaches imitating strong partitioning.

Table 6: Statistics on the effective optimality gap (9) after Alpine's first iteration (note: the minimum possible value of the effective optimality gap is 10^{-4}, the target gap). Columns show the geometric mean, median, minimum, and maximum effective gaps over 1000 instances. The numbers in these columns have been multiplied by 10^4 for ease of readability. The last column indicates the percentage of instances for which each method achieves the minimum possible effective optimality gap of 10^{-4} after Alpine's first iteration.

Problem Family	Solution Method	$10^4\times$ Effective Optimality Gap				% Instances
		GM	Median	Min	Max	Gap Closed
Bilinear, $n = 10$	Alpine (default)	5.5	4.5	1	343	0.1
	Alpine+SP2	1	1	1	1	100
	Alpine+ML2	1.2	1	1	439	88.3
Bilinear, $n = 20$	Alpine (default)	2869	3324	734	4805	0
	Alpine+SP2	1.3	1	1	61	82.3
	Alpine+ML2	16	19	1	1421	18.9
	Alpine+SP4	1	1	1	4.7	96.0
	Alpine+ML4	22	36	1	994	14.5
Bilinear, $n = 50$	Alpine (default)	144	170	1	693	0.1
	Alpine+SP2	1.7	1.2	1	5371	46.0
	Alpine+ML2	9.5	9.4	1	4941	5.6
QCQP, $n = 10$	Alpine (default)	13	12	7.5	187	0
	Alpine+SP2	1	1	1	1	100
	Alpine+ML2	3	1	1	1291	71.8
QCQP, $n = 20$	Alpine (default)	627	784	30	2064	0
	Alpine+SP2	2.1	1	1	66	52.2
	Alpine+ML2	20	25	1	578	2.0
	Alpine+SP4	1.1	1	1	36	92.6
	Alpine+ML4	15	17	1	668	14.7
QCQP, $n = 50$	Alpine (default)	81	104	6.3	282	0
	Alpine+SP2	1.6	1.3	1	10	39.0
	Alpine+ML2	4.8	5.3	1	148	14.9
Pooling	Alpine (default)	68	64	12	440	0
	Alpine+SP2	2.4	1.4	1	31	45.2
	Alpine+ML2	15	16	1	63	0.1

7 Future work

There are several promising avenues for future research. One potential direction is to move beyond the current approach of prespecifying a fixed number of partitioning points per variable in the strong partitioning problem (SP). Instead, we could allocate different numbers of partitioning points to each variable based on their relative impact on the lower bound. For example, consider a scenario where we aim to allocate up to $d + 2$ partitioning points per partitioned variable and have a total budget $B \in [d \times |\mathcal{NC}|]$ for unfixed partitioning points across all variables (excluding variable bounds). To optimally allocate and specify these partitioning points, we can solve the following max-min problem:

$$\max\left\{\underline{v}(P) \;:\; (P,Z) \in \mathcal{P}_d \times \{0,1\}^{|\mathcal{NC}|\times d},\ \sum_{i\in\mathcal{NC}}\sum_{j\in[d]} Z_{ij} = B,\ Z_{ij} = 0 \implies P_{i(j+1)} = 0,\ \forall (i,j) \in \mathcal{NC}\times[d]\right\},$$

Table 7: Statistics on the time required to run Algorithm 3 to determine strong partitioning points. Columns denote the shifted geometric mean, median, minimum, maximum, and standard deviation of these times.

Problem Family	Solution Method	Max-Min Solution Time (seconds)				
		Shifted GM	Median	Min	Max	Std. Deviation
Bilinear, $n = 10$	SP2	16	14	6	96	13
Bilinear, $n = 20$	SP2	528	445	136	2389	544
	SP4	1244	1117	374	4360	893
Bilinear, $n = 50$	SP2	7070	7404	1271	23166	3268
QCQP, $n = 10$	SP2	8	8	6	53	3
QCQP, $n = 20$	SP2	1731	1826	171	4244	654
	SP4	2152	2740	471	5965	961
QCQP, $n = 50$	SP2	16964	17074	8626	23551	2319
Pooling	SP2	15658	15148	1088	77029	8657

where $P \in [0,1]^{|\mathcal{NC}|\times(d+2)}$ represents the potential partitioning points and $\underline{v}$ is the optimal value function of problem (PMR-OA). Here, if $Z_{ij} = 0$, the partitioning point $P_{i(j+1)}$ is forced to 0, rendering it redundant. Unlike the original strong partitioning problem (SP), the outer-maximization problem above involves binary decision variables Z, which necessitates new techniques for its solution.

Additionally, future work could focus on designing more efficient methods for solving the max-min problem (SP) to enhance the scalability of generating strong partitioning expert data. Another exciting direction is designing tailored ML architectures capable of achieving similar speedups as the SP policy while handling varying feature and output dimensions. Finally, it would be interesting to explore generalizations of the SP policy that optimally select a subset of variables for partitioning at each iteration of Algorithm 1.

Acknowledgments

We gratefully acknowledge funding from Los Alamos National Laboratory's Center for Nonlinear Studies and the U.S. Department of Energy's LDRD program under the projects "20230091ER: Learning to Accelerate Global Solutions for Non-convex Optimization" and "20210078DR: The Optimization of Machine Learning: Imposing Requirements on Artificial Intelligence." This research used resources provided by LANL's Darwin testbed, which is funded by the Computational Systems and Software Environments subprogram of LANL's Advanced Simulation and Computing program (NNSA/DOE).

References

[1] T. Achterberg. *Constraint integer programming.* PhD thesis, TU Berlin, 2007.

[2] F. A. Al-Khayyal and J. E. Falk. Jointly constrained biconvex programming. *Mathematics of Operations Research*, 8(2): 273–286, 1983.

[3] A. M. Alvarez, Q. Louveaux, and L. Wehenkel. A machine learning-based approximation of strong branching. *INFORMS Journal on Computing*, 29(1):185–195, 2017.

[4] R. Baltean-Lugojan, P. Bonami, R. Misener, and A. Tramontani. Scoring positive semidefinite cutting planes for quadratic optimization via trained neural networks. *Optimization Online*, 2019. URL https://optimization-online.org/2018/11/6943/.

[5] X. Bao, N. V. Sahinidis, and M. Tawarmalani. Semidefinite relaxations for quadratically constrained quadratic programming: A review and comparisons. *Mathematical Programming*, 129(1):129–157, 2011.

[6] Y. Bengio, A. Lodi, and A. Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

[7] M. L. Bergamini, I. Grossmann, N. Scenna, and P. Aguirre. An improved piecewise outer-approximation algorithm for the

global optimization of MINLP models involving concave and bilinear terms. *Computers & Chemical Engineering*, 32(3): 477–493, 2008.

[8] K. Bestuzheva, M. Besançon, W.-K. Chen, et al. The SCIP optimization suite 8.0. *arXiv preprint: 2112.08872*, 2021.

[9] D. Bienstock, M. Escobar, C. Gentile, and L. Liberti. Mathematical programming formulations for the alternating current optimal power flow problem. *Annals of Operations Research*, pages 1–39, 2022.

[10] A. Billionnet, S. Elloumi, and A. Lambert. Extending the QCR method to general mixed-integer programs. *Mathematical Programming*, 131(1):381–401, 2012.

[11] P. Bonami, A. Lodi, and G. Zarpellon. Learning a classification of mixed-integer quadratic programming problems. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 595–604. Springer, 2018.

[12] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and regression trees*. Routledge, 2017.

[13] S. Burer and D. Vandenbussche. A finite branch-and-bound algorithm for nonconvex quadratic programming via semidefinite relaxations. *Mathematical Programming*, 113(2):259–282, 2008.

[14] Q. Cappart, D. Chételat, E. B. Khalil, A. Lodi, C. Morris, and P. Veličković. Combinatorial optimization and reasoning with graph neural networks. *Journal of Machine Learning Research*, 24(130):1–61, 2023.

[15] P. M. Castro. Normalized multiparametric disaggregation: an efficient relaxation for mixed-integer bilinear problems. *Journal of Global Optimization*, 64(4):765–784, 2016.

[16] F. Cengil, H. Nagarajan, R. Bent, S. Eksioglu, and B. Eksioglu. Learning to accelerate globally optimal solutions to the AC optimal power flow problem. *Electric Power Systems Research*, 212:108275, 2022.

[17] D. De Wolf and Y. Smeers. Generalized derivatives of the optimal value of a linear program with respect to matrix coefficients. *European Journal of Operational Research*, 291(2):491–496, 2021.

[18] G. Di Liberto, S. Kadioglu, K. Leo, and Y. Malitsky. DASH: Dynamic approach for switching heuristics. *European Journal of Operational Research*, 248(3):943–953, 2016.

[19] H. Drucker. Improving regressors using boosting techniques. In *ICML*, volume 97, pages 107–115. Citeseer, 1997.

[20] R. M. Freund. Postoptimal analysis of a linear program under simultaneous changes in matrix coefficients. In *Mathematical Programming Essays in Honor of George B. Dantzig Part I*, pages 1–13. Springer, 1985.

[21] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[22] F. Furini, E. Traversi, P. Belotti, A. Frangioni, A. Gleixner, N. Gould, L. Liberti, A. Lodi, R. Misener, H. Mittelmann, et al. QPLIB: a library of quadratic programming instances. *Mathematical Programming Computation*, 11:237–265, 2019.

[23] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi. Exact combinatorial optimization with graph convolutional neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[24] B. Ghaddar, I. Gómez-Casares, J. González-Díaz, B. González-Rodríguez, B. Pateiro-López, and S. Rodríguez-Ballesteros. Learning for spatial branching: An algorithm selection approach. *INFORMS Journal on Computing*, 35(5):1024–1043, 2023.

[25] B. González-Rodríguez, R. Alvite-Pazó, S. Alvite-Pazó, B. Ghaddar, and J. González-Díaz. Polynomial optimization: Enhancing RLT relaxations with conic constraints. *arXiv preprint arXiv:2208.05608*, 2022.

[26] C. A. Haverly. Studies of the behavior of recursion for the pooling problem. *ACM SIGMAP Bulletin*, 25:19–28, 1978.

[27] H. He, H. Daume III, and J. M. Eisner. Learning to search in branch and bound algorithms. *Advances in Neural Information Processing Systems*, 27:3293–3301, 2014.

[28] J. Im. Sensitivity analysis and robust optimization: A geometric approach for the special case of linear optimization. Master's thesis, University of Waterloo, 2018.

[29] D. Jungen, A. Zingler, H. Djelassi, and A. Mitsos. libDIPS–Discretization-Based Semi-Infinite and Bilevel Programming Solvers. *Available on Optimization Online. URL: https://optimization-online.org/?p=24914*, pages 1–38, 2023.

[30] R. Kannan. *Algorithms, analysis and software for the global optimization of two-stage stochastic programs*. PhD thesis, Massachusetts Institute of Technology, 2018.

[31] R. Kannan and P. I. Barton. The cluster problem in constrained global optimization. *Journal of Global Optimization*, 69 (3):629–676, 2017.

[32] R. Kannan and P. I. Barton. Convergence-order analysis of branch-and-bound algorithms for constrained problems. *Journal of Global Optimization*, 71(4):753–813, 2018.

[33] E. Khalil, P. Le Bodic, L. Song, G. Nemhauser, and B. Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[34] J. Kim, J.-P. P. Richard, and M. Tawarmalani. Piecewise polyhedral relaxations of multilinear optimization. *Optimization Online*, 2022.

[35] T. C. Koopmans and M. Beckmann. Assignment problems and the location of economic activities. *Econometrica: Journal of the Econometric Society*, pages 53–76, 1957.

[36] M. Lee, N. Ma, G. Yu, and H. Dai. Accelerating generalized Benders decomposition for wireless resource allocation. *IEEE Transactions on Wireless Communications*, pages 1233–1247, 2020.

[37] X. Li, E. Armagan, A. Tomasgard, and P. I. Barton. Stochastic pooling problem for natural gas production network design and operation under uncertainty. *AIChE Journal*, 57(8):2120–2135, 2011.

[38] J. Liu, N. Ploskas, and N. V. Sahinidis. Tuning BARON using derivative-free optimization algorithms. *Journal of Global Optimization*, 74(4):611–637, 2019.

[39] A. Lodi and G. Zarpellon. On learning and branching: a survey. *Top*, 25(2):207–236, 2017.

[40] M. Lu, H. Nagarajan, R. Bent, S. D. Eksioglu, and S. J. Mason. Tight piecewise convex relaxations for global optimization of optimal power flow. In *2018 Power Systems Computation Conference*, pages 1–7. IEEE, 2018.

[41] J. Luedtke, C. d'Ambrosio, J. Linderoth, and J. Schweiger. Strong convex nonlinear relaxations of the pooling problem. *SIAM Journal on Optimization*, 30(2):1582–1609, 2020.

[42] M. M. Mäkelä. Multiobjective proximal bundle method for nonconvex nonsmooth optimization: Fortran subroutine MPBNGC 2.0. *Reports of the Department of Mathematical Information Technology, Series B. Scientific Computing, B*, 13, 2003.

[43] G. P. McCormick. Computability of global solutions to factorable nonconvex programs: Part I—convex underestimating problems. *Mathematical Programming*, 10(1):147–175, 1976.

[44] R. Misener and C. A. Floudas. GloMIQO: Global mixed-integer quadratic optimizer. *Journal of Global Optimization*, 57 (1):3–50, 2013.

[45] R. Misener and C. A. Floudas. ANTIGONE: algorithms for continuous/integer global optimization of nonlinear equations. *Journal of Global Optimization*, 59(2):503–526, 2014.

[46] H. Nagarajan, M. Lu, E. Yamangil, and R. Bent. Tightening McCormick relaxations for nonlinear programs via dynamic multivariate partitioning. In *International Conference on Principles and Practice of Constraint Programming*, pages 369–387. Springer, 2016.

[47] H. Nagarajan, M. Lu, S. Wang, R. Bent, and K. Sundar. An adaptive, multivariate partitioning algorithm for global optimization of nonconvex programs. *Journal of Global Optimization*, 74(4):639–675, 2019.

[48] V. Nair, S. Bartunov, F. Gimeno, et al. Solving mixed integer programs using neural networks. *arXiv preprint: 2012.13349*, 2020.

[49] G. Nannicini, P. Belotti, J. Lee, J. Linderoth, F. Margot, and A. Wächter. A probing algorithm for MINLP with failure prediction by SVM. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 154–169. Springer, 2011.

[50] C. J. Nohra, A. U. Raghunathan, and N. Sahinidis. Spectral relaxations and branching strategies for global optimization of mixed-integer quadratic programs. *SIAM Journal on Optimization*, 31(1):142–171, 2021.

[51] C. J. Nohra, A. U. Raghunathan, and N. V. Sahinidis. SDP-quality bounds via convex quadratic relaxations for global optimization of mixed-integer quadratic programs. *Mathematical Programming*, 196(1):203–233, 2022.

[52] P. M. Pardalos and S. A. Vavasis. Quadratic programming with one negative eigenvalue is NP-hard. *Journal of Global Optimization*, 1(1):15–22, 1991.

[53] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, ..., and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[54] N. V. Sahinidis. BARON: A general purpose global optimization software package. *Journal of Global Optimization*, 8(2): 201–205, 1996.

[55] Y. Saif, A. Elkamel, and M. Pritzker. Global optimization of reverse osmosis network for wastewater treatment and minimization. *Industrial & Engineering Chemistry Research*, 47(9):3060–3070, 2008.

[56] H. D. Sherali and W. P. Adams. *A reformulation-linearization technique for solving discrete and continuous nonconvex problems*, volume 31. Springer Science & Business Media, 2013.

[57] N. Z. Shor. Quadratic optimization problems. *Soviet Journal of Computer and Systems Sciences*, 25:1–11, 1987.

[58] G. Still. Lectures on parametric optimization: An introduction. *Optimization Online*, 2018.

[59] K. Sundar, H. Nagarajan, J. Linderoth, S. Wang, and R. Bent. Piecewise polyhedral formulations for a multilinear term. *Operations Research Letters*, 49(1):144–149, 2021.

[60] D. S. Wicaksono and I. A. Karimi. Piecewise MILP under- and overestimators for global optimization of bilinear programs. *AIChE Journal*, 54(4):991–1008, 2008.

[61] Y. Yang and P. I. Barton. Integrated crude selection and refinery optimization under uncertainty. *AIChE journal*, 62(4): 1038–1053, 2016.