Learning to Accelerate Partitioning Algorithms for the Global Optimization of Nonconvex Quadratically-Constrained Quadratic Programs

Rohit Kannan

Center for Nonlinear Studies
Los Alamos National Laboratory

November 13, 2022

Joint work with Deepjyoti Deka and Harsha Nagarajan (LANL)

Funding

1. LANL LDRD 20230091ER: "Learning to Accelerate Global Solutions for Non-convex Optimization"
2. Center for Nonlinear Studies at LANL

Motivation

Many important applications can be formulated as nonconvex QCQPs

AC Optimal Power Flow

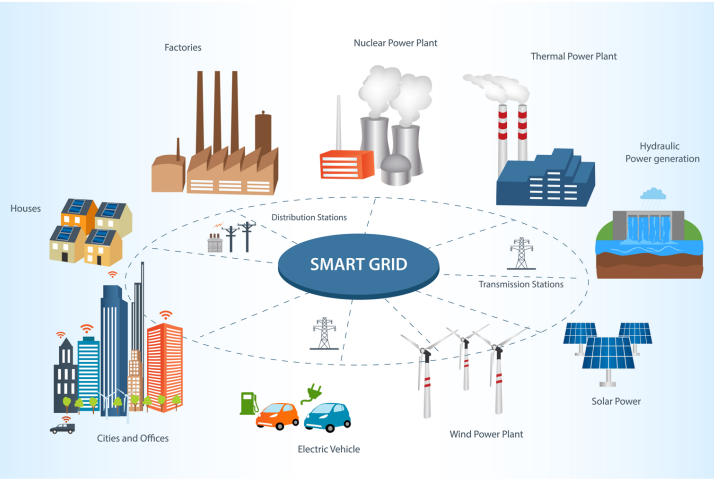
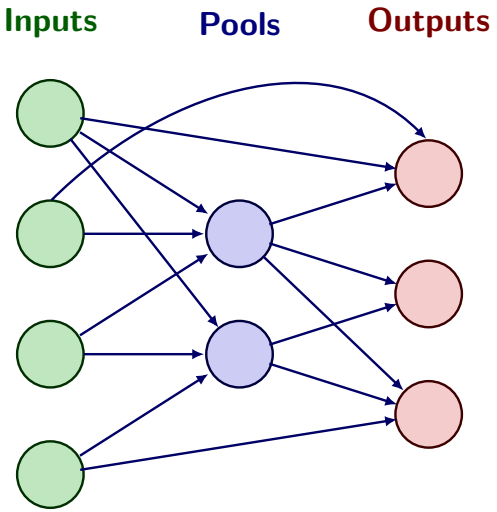


Image Source: IEEE Innovation at Work

The Pooling Problem



Motivation

Many important applications can be formulated as nonconvex QCQPs

AC Optimal Power Flow

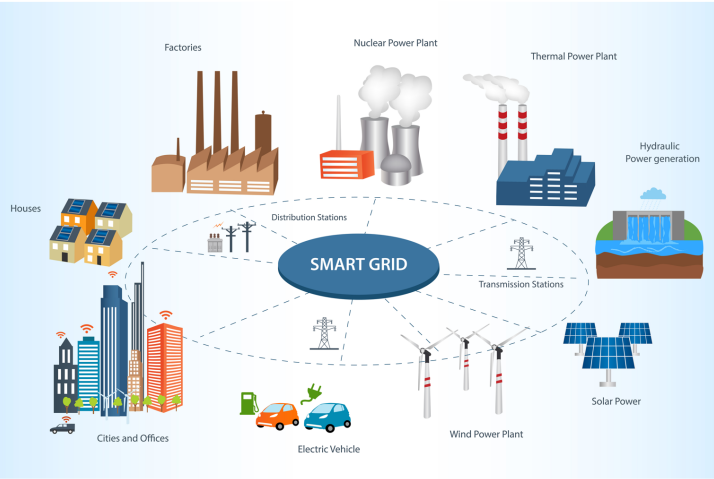
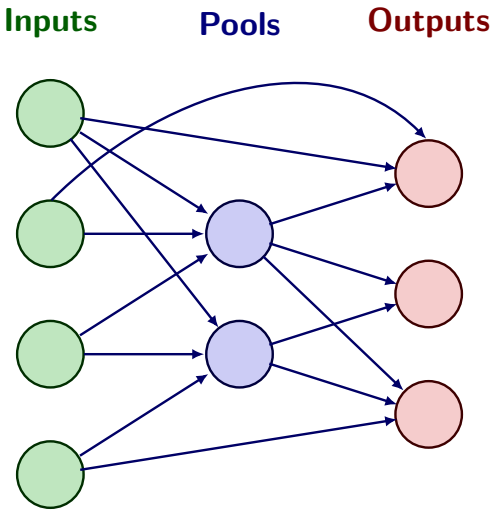


Image Source: IEEE Innovation at Work

The Pooling Problem



Often, wish to *repeatedly* solve instances of the same nonconvex problem with different data, e.g., loads, wind, qualities, prices

Motivation

Many important applications can be formulated as nonconvex QCQPs

AC Optimal Power Flow

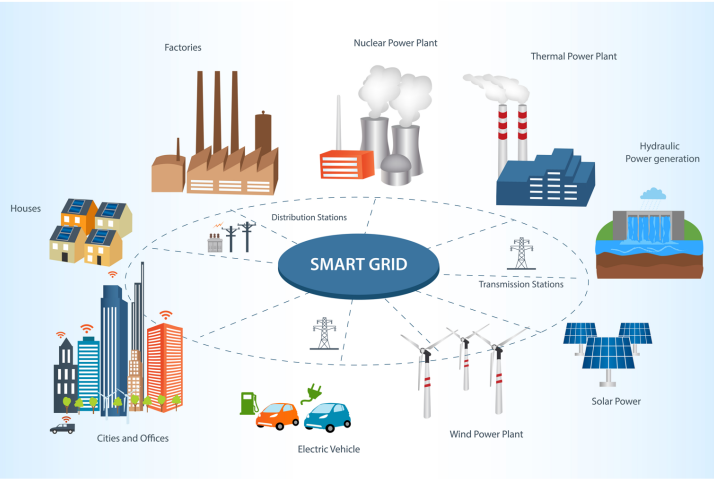
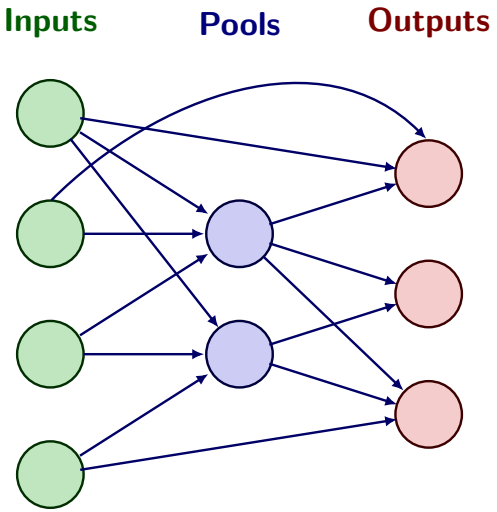


Image Source: IEEE Innovation at Work

The Pooling Problem



Often, wish to *repeatedly* solve instances of the same nonconvex problem with different data, e.g., loads, wind, qualities, prices

Can we exploit *shared structure* to accelerate global solution?

Heuristics can have a huge impact on global solvers

`Alpine` [NLW+19] is a Julia-based open-source global solver developed at LANL for mixed-integer polynomial optimization

Heuristics can have a huge impact on global solvers

`Alpine` [NLW+19] is a Julia-based open-source global solver developed at LANL for mixed-integer polynomial optimization

It has a key algorithmic parameter "PSF" (default PSF $= 10$)

PSF = Partition Scaling Factor. More on this parameter later...

Heuristics can have a huge impact on global solvers

`Alpine` [NLW+19] is a Julia-based open-source global solver developed at LANL for mixed-integer polynomial optimization

It has a key algorithmic parameter "PSF" (default PSF = 10)

PSF = Partition Scaling Factor. More on this parameter later...

Best choice of PSF can vary depending on instance

Alpine on three random QCQPs

PSF	**4**	**10**	**15**
Time for Ex1:	5087s	**704s**	1551s
Time for Ex2:	**2632s**	5023s	6642s
Time for Ex3:	3000s	4540s	**1433s**

Note: Alpine converges irrespective of the choice of PSF

Heuristics can have a huge impact on global solvers

`Alpine` [NLW+19] is a Julia-based open-source global solver developed at LANL for mixed-integer polynomial optimization

It has a key algorithmic parameter "PSF" (default PSF = 10)

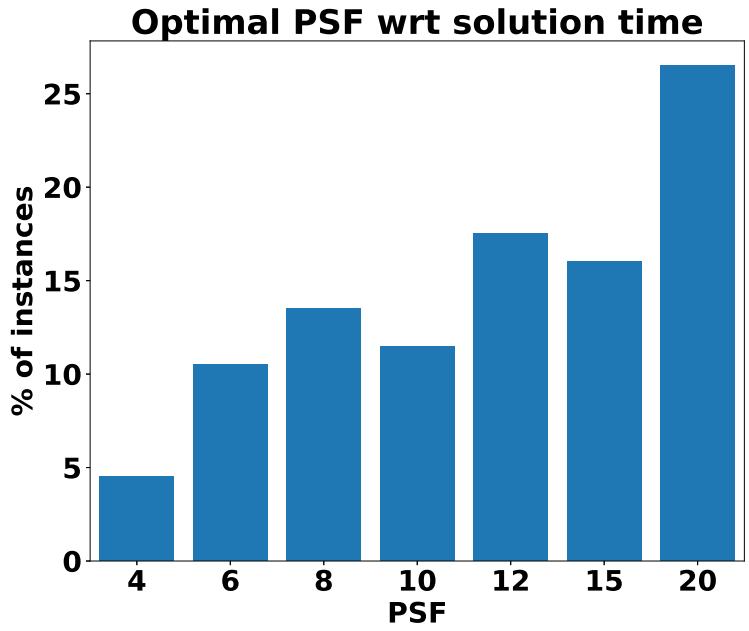
PSF = Partition Scaling Factor. More on this parameter later...

Best choice of PSF can vary depending on instance

Alpine on three random QCQPs

PSF	4	10	15
Time for Ex1:	5087s	**704s**	1551s
Time for Ex2:	**2632s**	5023s	6642s
Time for Ex3:	3000s	4540s	**1433s**

Note: Alpine converges irrespective of the choice of PSF



Heuristics can have a huge impact on global solvers

`Alpine` [NLW+19] is a Julia-based open-source global solver developed at LANL for mixed-integer polynomial optimization

It has a key algorithmic parameter "PSF" (default PSF = 10)

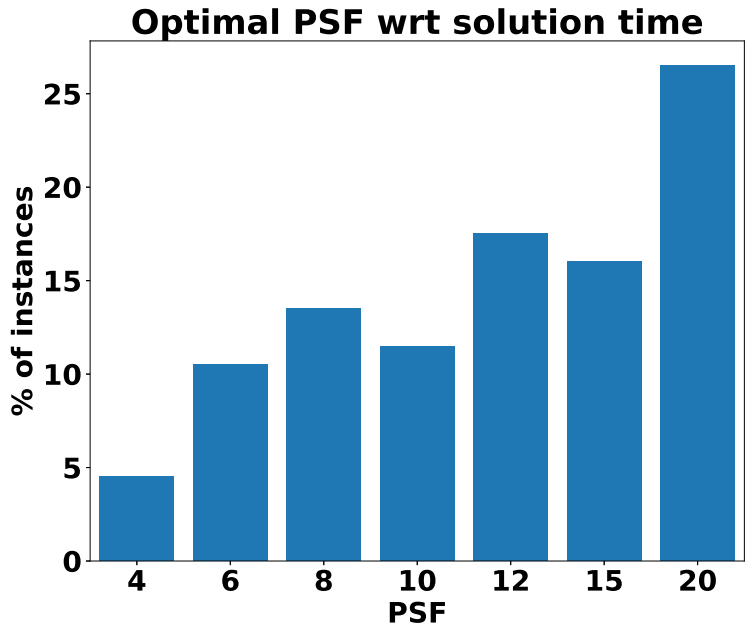
PSF = Partition Scaling Factor. More on this parameter later...

Best choice of PSF can vary depending on instance

Alpine on three random QCQPs

PSF	4	10	15
Time for Ex1:	5087s	**704s**	1551s
Time for Ex2:	**2632s**	5023s	6642s
Time for Ex3:	3000s	4540s	**1433s**

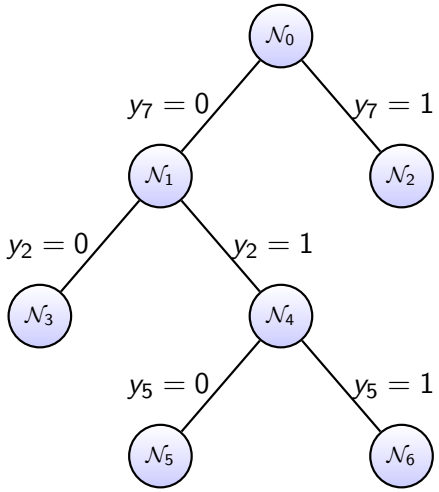
Note: Alpine converges irrespective of the choice of PSF



In general, how to optimally specify a solver's heuristic parameters *for a given instance*? Heuristics usually tuned to work well *on average*

Related Work: Learning to Branch for MIPs

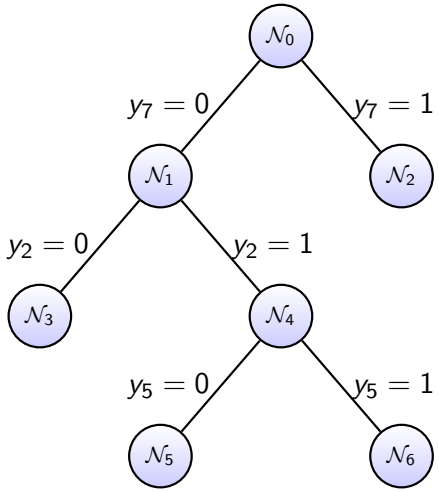
$$\text{(MIP)} \quad \min_{x,y} \; c^\mathsf{T} x + d^\mathsf{T} y$$
$$\text{s.t.} \; Ax + By \leq b,$$
$$x \geq 0, \;\; y \in \{0,1\}^{d_y}$$



- Order of branching decisions can be critical to ensuring that the size of the branch-and-bound tree doesn't explode
  How to choose branching order?

Related Work: Learning to Branch for MIPs

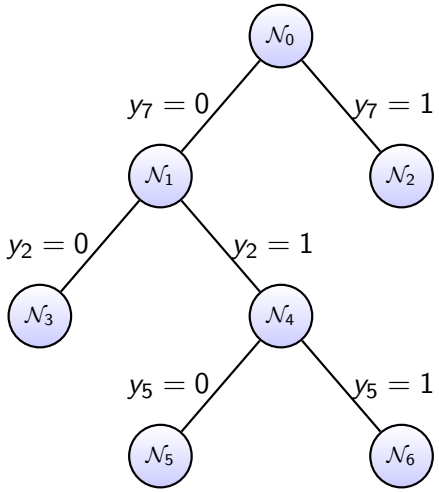
$$\text{(MIP)} \quad \min_{x,y} \; c^\top x + d^\top y$$
$$\text{s.t.} \; Ax + By \leq b,$$
$$x \geq 0, \;\; y \in \{0,1\}^{d_y}$$



- Order of branching decisions can be critical to ensuring that the size of the branch-and-bound tree doesn't explode
  How to choose branching order?
- **Strong Branching** to choose branching variable at node
  1. Try branching on all candidate y's, e.g. at $\mathcal{N}_4$: y_1, y_4, y_5, y_{10}
  2. Branch on y_{i^*} to maximize the lower bound on the child nodes

Related Work: Learning to Branch for MIPs

$$\text{(MIP)} \quad \min_{x,y} \; c^\top x + d^\top y$$
$$\text{s.t.} \; Ax + By \leq b,$$
$$x \geq 0, \;\; y \in \{0,1\}^{d_y}$$

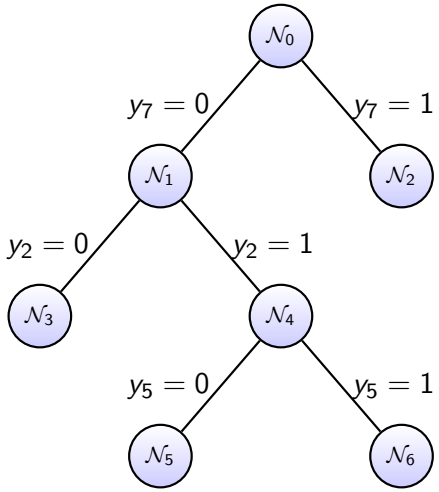


- Order of branching decisions can be critical to ensuring that the size of the branch-and-bound tree doesn't explode
  How to choose branching order?
- **Strong Branching** to choose branching variable at node
  1. Try branching on all candidate y's, e.g. at $\mathcal{N}_4$: y_1, y_4, y_5, y_{10}
  2. Branch on y_{i^*} to maximize the lower bound on the child nodes

  Empirically reduces number of nodes in B&B tree by 65% on average, but increases cost per node by 44%

Related Work: Learning to Branch for MIPs

$$\text{(MIP)} \quad \min_{x,y} \; c^\top x + d^\top y$$
$$\text{s.t.} \; Ax + By \leq b,$$
$$x \geq 0, \;\; y \in \{0,1\}^{d_y}$$



- Order of branching decisions can be critical to ensuring that the size of the branch-and-bound tree doesn't explode
  How to choose branching order?
- **Strong Branching** to choose branching variable at node
  1. Try branching on all candidate y's, e.g. at $\mathcal{N}_4$: y_1, y_4, y_5, y_{10}
  2. Branch on y_{i^*} to maximize the lower bound on the child nodes

  Empirically reduces number of nodes in B&B tree by 65% on average, but increases cost per node by 44%
- Several recent works use ML to compute cheap proxy of strong branching for MILPs [ALW17, KLBS+16, GCF+19, NBG+20]

Related Work: Learning for (MI)NLPs

- Baltean-Lugojan et al. [BLBMT19] use neural networks (NNs) to decide how to construct cheap outer-approximations of SDP relaxations of QCQPs that retain their strength
- Ghaddar et al. [GGCGD+22, GRAPAP+22] use quantile regression forests to choose a branching strategy within the reformulation-linearization technique for polynomial programs

Related Work: Learning for (MI)NLPs

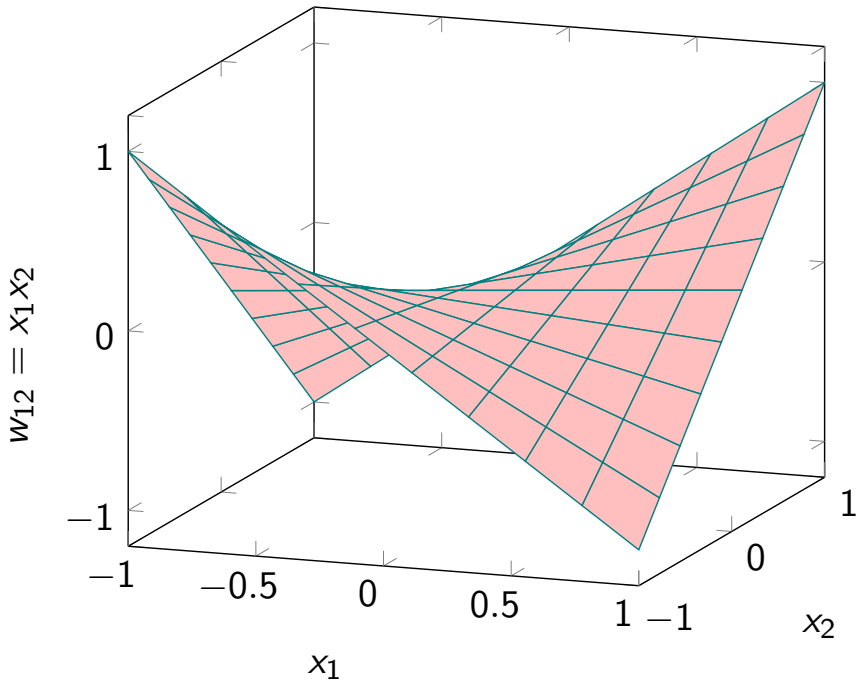
- Baltean-Lugojan et al. [BLBMT19] use neural networks (NNs) to decide how to construct cheap outer-approximations of SDP relaxations of QCQPs that retain their strength
- Ghaddar et al. [GGCGD+22, GRAPAP+22] use quantile regression forests to choose a branching strategy within the reformulation-linearization technique for polynomial programs
- Bonami et al. [BLZ18] learn a classifier to decide whether to linearize binary-binary or binary-continuous products in MIQPs
- Nannicini et al. [NBL+11] train an SVM classifier to predict whether to use an expensive bound tightening procedure instead of feasibility-based bound tightening for MINLPs
- Cengil et al. [CNB+22] train DNNs to choose a subset of variables on which to apply optimality-based bounds tightening for AC Optimal Power Flow

Global Optimization of QCQPs

Consider the following class of QCQPs:

$$\nu^* := \min_{x,w} \; c^\mathsf{T} x + d^\mathsf{T} w$$
$$\text{s.t. } w_{ij} = x_i x_j, \quad \forall (i,j) \in \mathcal{B},$$
$$Ax + Bw \leq b, \;\; x \in [-1,1]^{d_x}$$

- The bilinear constraints are what make the problem hard



Global Optimization of QCQPs

Consider the following class of QCQPs:

$$\begin{aligned} \nu^* := \min_{x,w} \; & c^\mathsf{T} x + d^\mathsf{T} w \\ \text{s.t.} \; & w_{ij} = x_i x_j, \quad \forall (i,j) \in \mathcal{B}, \\ & Ax + Bw \leq b, \;\; x \in [-1,1]^{d_x} \end{aligned}$$

- The bilinear constraints are what make the problem hard
- Get feasible solutions/upper bounds using local optimization
- Obtain lower bounds on ν^* using relaxations

Relaxing Bilinear Terms

The feasible region of the hard *bilinear* constraints

$$w_{ij} = x_i x_j, \quad x_i, x_j \in [-1, 1] \tag{1}$$

is a subset of the feasible region of the easy *linear* constraints

$$\begin{aligned} -x_i - x_j - 1 \leq w_{ij} \leq x_i - x_j + 1, \\ x_i + x_j - 1 \leq w_{ij} \leq x_j - x_i + 1, \\ x_i, x_j \in [-1, 1] \end{aligned} \tag{2}$$

Relaxing Bilinear Terms

The feasible region of the hard *bilinear* constraints

$$w_{ij} = x_i x_j, \quad x_i, x_j \in [-1, 1] \tag{1}$$

is a subset of the feasible region of the easy *linear* constraints

$$\begin{aligned} -x_i - x_j - 1 \leq w_{ij} \leq x_i - x_j + 1, \\ x_i + x_j - 1 \leq w_{ij} \leq x_j - x_i + 1, \\ x_i, x_j \in [-1, 1] \end{aligned} \tag{2}$$

Replace bilinear constraints (1) in the QCQP with McCormick Relaxations (2) to determine a valid lower bound

$$\begin{aligned} \nu^* \geq \nu^M := \min_{x,w} \; & c^\mathsf{T} x + d^\mathsf{T} w \\ \text{s.t.} \; & Ax + Bw \leq b, \\ & -x_i - x_j - 1 \leq w_{ij} \leq x_i - x_j + 1, \quad \forall (i,j) \in \mathcal{B}, \\ & x_i + x_j - 1 \leq w_{ij} \leq x_j - x_i + 1, \quad \forall (i,j) \in \mathcal{B}, \\ & x \in [-1, 1]^{d_x} \end{aligned}$$

Relaxing Bilinear Terms

The feasible region of the hard *bilinear* constraints

$$w_{ij} = x_i x_j, \quad x_i, x_j \in [-1, 1] \tag{1}$$

is a subset of the feasible region of the easy *linear* constraints

$$\begin{aligned} -x_i - x_j - 1 \leq w_{ij} \leq x_i - x_j + 1, \\ x_i + x_j - 1 \leq w_{ij} \leq x_j - x_i + 1, \\ x_i, x_j \in [-1, 1] \end{aligned} \tag{2}$$

Replace bilinear constraints (1) in the QCQP with McCormick Relaxations (2) to determine a valid lower bound

$$\begin{aligned} \nu^* \geq \nu^M := \min_{x,w} \; & c^\mathsf{T} x + d^\mathsf{T} w \\ \text{s.t.} \; & Ax + Bw \leq b, \\ & -x_i - x_j - 1 \leq w_{ij} \leq x_i - x_j + 1, \quad \forall (i,j) \in \mathcal{B}, \\ & x_i + x_j - 1 \leq w_{ij} \leq x_j - x_i + 1, \quad \forall (i,j) \in \mathcal{B}, \\ & x \in [-1, 1]^{d_x} \end{aligned}$$

Typically $\nu^M \ll \nu^*$. Close the gap using continuous B&B

Tighten Relaxations By Partitioning Variable Domains

- Partition variable domains into "disjoint" subintervals, e.g.,

$$x_1 \in [-1, -0.5] \text{ OR } [-0.5, 0] \text{ OR } [0, 1]$$
$$x_2 \in [-1, -0.5] \text{ OR } [-0.5, 1]$$

Tighten Relaxations By Partitioning Variable Domains

- Partition variable domains into "disjoint" subintervals, e.g.,

$$x_1 \in [-1, -0.5] \text{ OR } [-0.5, 0] \text{ OR } [0, 1]$$
$$x_2 \in [-1, -0.5] \text{ OR } [-0.5, 1]$$

- Construct Piecewise McCormick Relaxations on the variable partitions and solve a MIP to obtain lower bound

$$\begin{aligned} \nu^* \geq \nu^{PMR} := \min_{x,w} \; & c^\top x + d^\top w \\ \text{s.t.} \; & Ax + Bw \leq b, \\ & (x_i, x_j, w_{ij}) \in \mathcal{PMR}_{ij}(p_i, p_j), \quad \forall (i,j) \in \mathcal{B}, \\ & x \in [-1, 1]^{d_x}, \end{aligned}$$

where p_i is the vector of partitioning points for x_i

Tighten Relaxations By Partitioning Variable Domains

- Partition variable domains into "disjoint" subintervals, e.g.,

$$x_1 \in [-1, -0.5] \text{ OR } [-0.5, 0] \text{ OR } [0, 1]$$
$$x_2 \in [-1, -0.5] \text{ OR } [-0.5, 1]$$

- Construct Piecewise McCormick Relaxations on the variable partitions and solve a MIP to obtain lower bound

$$\begin{aligned} \nu^* \geq \nu^{PMR} := \min_{x,w} \; & c^\top x + d^\top w \\ \text{s.t. } & Ax + Bw \leq b, \\ & (x_i, x_j, w_{ij}) \in \mathcal{PMR}_{ij}(p_i, p_j), \quad \forall (i,j) \in \mathcal{B}, \\ & x \in [-1, 1]^{d_x}, \end{aligned}$$

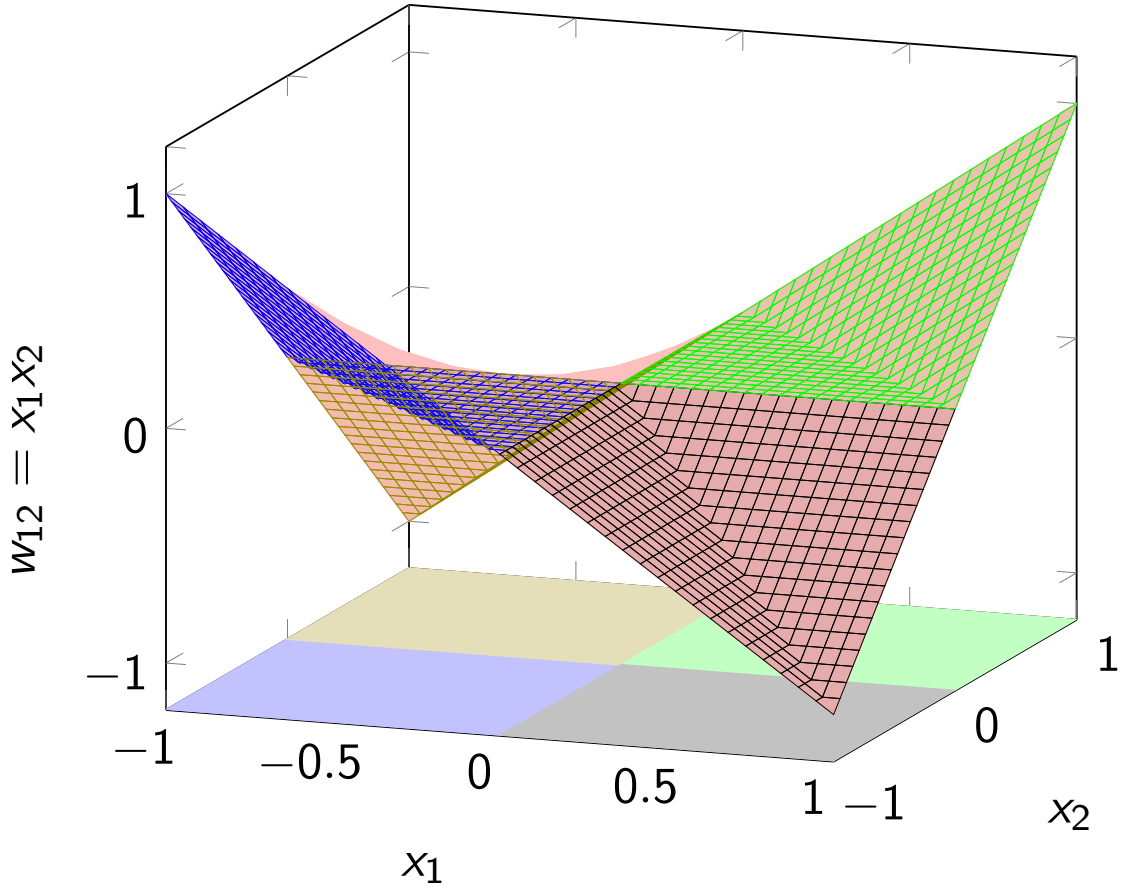
where p_i is the vector of partitioning points for x_i

- Refine variable partitions, e.g.,

$$x_1 \in [-1, -0.5] \text{ OR } [-0.5, 0] \quad \text{OR } [0, 1]$$
$$x_2 \in [-1, -0.5] \text{ OR } [-0.5, -0.2] \text{ OR } [-0.2, 1]$$

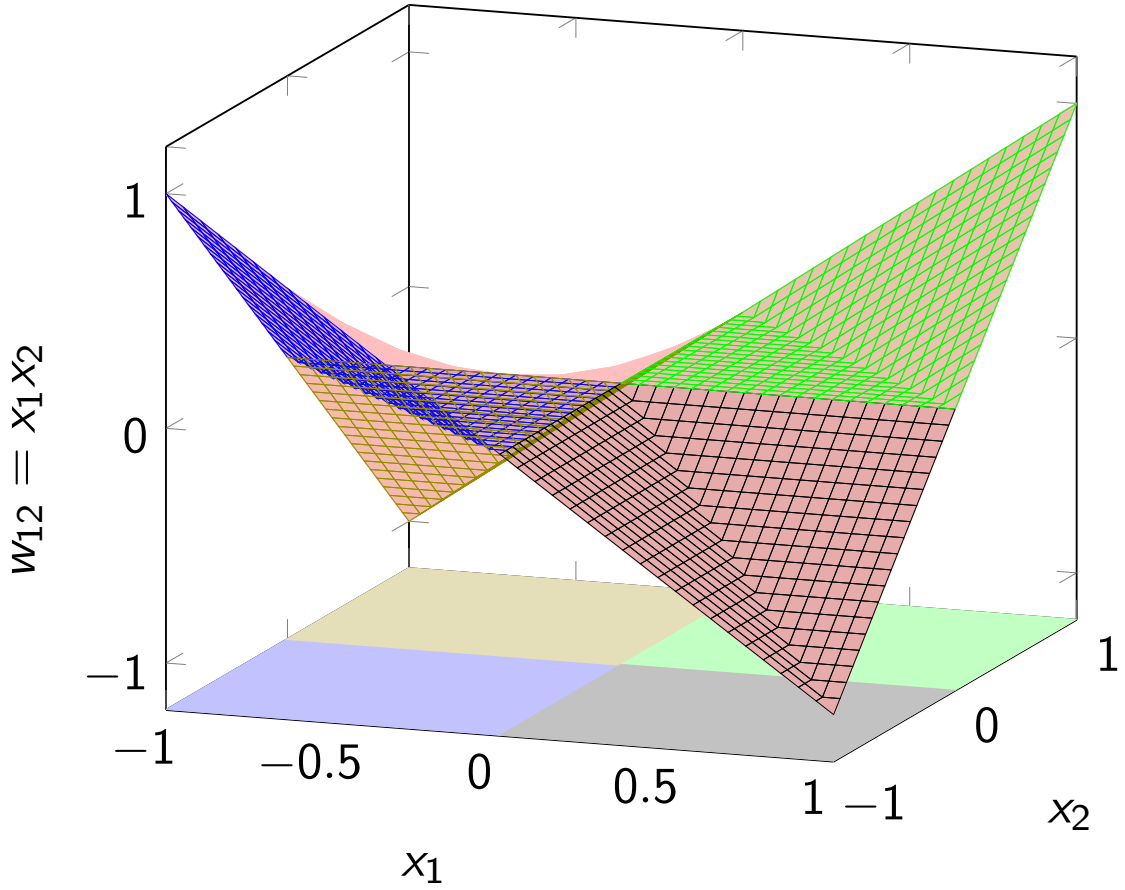
The Lower Part of the Piecewise McCormick Relaxations

Partitions: $x_1 \in [-1, 0]$ OR $[0, 1]$, $\quad x_2 \in [-1, 0]$ OR $[0, 1]$



The Lower Part of the Piecewise McCormick Relaxations

Partitions: $x_1 \in [-1, 0]$ OR $[0, 1]$, $\quad x_2 \in [-1, 0]$ OR $[0, 1]$



Both the number AND choice of partitioning points influence number of iterations for Alpine to converge

How Does Alpine Pick Partitioning Points?

Recall: `Alpine` has a key algorithmic parameter "PSF" (default PSF $= 10$)

Best choice of PSF can vary depending on instance

PSF	**4**	**10**	**15**
Time for Ex1:	5087s	**704s**	1551s
Time for Ex2:	**2632s**	5023s	6642s
Time for Ex3:	3000s	4540s	**1433s**

How Does Alpine Pick Partitioning Points?

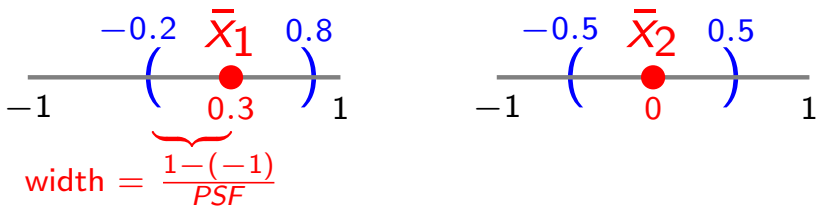
Recall: `Alpine` has a key algorithmic parameter "PSF" (default PSF $= 10$)

Best choice of PSF can vary depending on instance

PSF	**4**	**10**	**15**
Time for Ex1:	5087s	**704s**	1551s
Time for Ex2:	**2632s**	5023s	6642s
Time for Ex3:	3000s	4540s	**1433s**

Alpine's strategy: refine partitions around a nominal point $\bar{x}$ (e.g., around a feasible solution or solution to relaxation)

- Example: if $\bar{x} = (0.3, 0)$ and parameter PSF $= 4$



How Does Alpine Pick Partitioning Points?

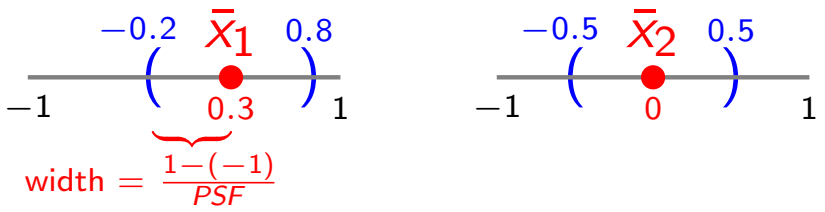
Recall: `Alpine` has a key algorithmic parameter "PSF" (default PSF $= 10$)

Best choice of PSF can vary depending on instance

PSF	**4**	**10**	**15**
Time for Ex1:	5087s	**704s**	1551s
Time for Ex2:	**2632s**	5023s	6642s
Time for Ex3:	3000s	4540s	**1433s**

Alpine's strategy: refine partitions around a nominal point $\bar{x}$ (e.g., around a feasible solution or solution to relaxation)

- Example: if $\bar{x} = (0.3, 0)$ and parameter PSF $= 4$



Although there are some empirical and theoretical motivations for the above partitioning strategy, it is still quite ad hoc

Can we choose better partitioning points to promote faster convergence?

Strong Partitioning (SP) to Improve Choice of Partitions

New Approach: Choose partitioning points to maximize the lower bound

$$p^* \in \arg\max_{p \in P} \nu^{PMR}(p),$$

- p_i is the vector of partitioning points for x_i

Strong Partitioning (SP) to Improve Choice of Partitions

New Approach: Choose partitioning points to maximize the lower bound

$$p^* \in \arg\max_{p \in P} \nu^{PMR}(p),$$

- p_i is the vector of partitioning points for x_i

$$\begin{aligned} \nu^{PMR}(p) := \min_{x,w} \; & c^\top x + d^\top w \\ \text{s.t.} \; & Ax + Bw \leq b, \\ & (x_i, x_j, w_{ij}) \in \mathcal{PMR}_{ij}(p_i, p_j), \quad \forall (i,j) \in \mathcal{B}, \\ & x \in [-1, 1]^{d_x}, \end{aligned}$$

- From iteration 2, use Alpine's partitioning strategy (guaranteed to converge irrespective of points chosen by SP)

Strong Partitioning (SP) to Improve Choice of Partitions

New Approach: Choose partitioning points to maximize the lower bound

$$p^* \in \arg\max_{p \in P} \nu^{PMR}(p),$$

- p_i is the vector of partitioning points for x_i

$$\begin{aligned} \nu^{PMR}(p) := \min_{x,w} \; & c^\top x + d^\top w \\ \text{s.t.} \; & Ax + Bw \leq b, \\ & (x_i, x_j, w_{ij}) \in \mathcal{PMR}_{ij}(p_i, p_j), \quad \forall (i,j) \in \mathcal{B}, \\ & x \in [-1, 1]^{d_x}, \end{aligned}$$

- From iteration 2, use Alpine's partitioning strategy (guaranteed to converge irrespective of points chosen by SP)

How to solve this max-min problem (locally)?

Strong Partitioning (SP) to Improve Choice of Partitions

New Approach: Choose partitioning points to maximize the lower bound

$$p^* \in \arg\max_{p \in P} \nu^{PMR}(p),$$

- p_i is the vector of partitioning points for x_i

$$\begin{aligned} \nu^{PMR}(p) := \min_{x,w} \; & c^\top x + d^\top w \\ \text{s.t.} \; & Ax + Bw \leq b, \\ & (x_i, x_j, w_{ij}) \in \mathcal{PMR}_{ij}(p_i, p_j), \quad \forall (i,j) \in \mathcal{B}, \\ & x \in [-1, 1]^{d_x}, \end{aligned}$$

- From iteration 2, use Alpine's partitioning strategy (guaranteed to converge irrespective of points chosen by SP)

How to solve this max-min problem (locally)? Using sensitivity analysis

Strong Partitioning (SP) to Improve Choice of Partitions

New Approach: Choose partitioning points to maximize the lower bound

$$p^* \in \arg\max_{p \in P} \nu^{PMR}(p),$$

- p_i is the vector of partitioning points for x_i

$$\begin{aligned} \nu^{PMR}(p) := \min_{x,w} \; & c^\top x + d^\top w \\ \text{s.t.} \; & Ax + Bw \leq b, \\ & (x_i, x_j, w_{ij}) \in \mathcal{PMR}_{ij}(p_i, p_j), \quad \forall (i,j) \in \mathcal{B}, \\ & x \in [-1, 1]^{d_x}, \end{aligned}$$

- From iteration 2, use Alpine's partitioning strategy (guaranteed to converge irrespective of points chosen by SP)

How to solve this max-min problem (locally)? Using sensitivity analysis

Solving this max-min problem may be as hard as solving the QCQP!

Using ML to Accelerate Alpine

Given family of random QCQPs of the form [BST09]

$$\begin{aligned} \nu^*(\theta) := \min_{x,w} \; & c(\theta)^\mathsf{T} x + d(\theta)^\mathsf{T} w \\ \text{s.t.} \; & A(\theta)x + B(\theta)w \leq b, \\ & w_{ij} = x_i x_j, \quad \forall (i,j) \in \mathcal{B}, \\ & x \in [0,1]^{d_x} \end{aligned}$$

Parameters θ vary from one instance to the next

Using ML to Accelerate Alpine

Given family of random QCQPs of the form [BST09]

$$\begin{aligned} \nu^*(\theta) := \min_{x,w} \; & c(\theta)^\mathsf{T} x + d(\theta)^\mathsf{T} w \\ \text{s.t.} \; & A(\theta)x + B(\theta)w \leq b, \\ & w_{ij} = x_i x_j, \quad \forall (i,j) \in \mathcal{B}, \\ & x \in [0,1]^{d_x} \end{aligned}$$

Parameters θ vary from one instance to the next

Input: underlying problem, distribution of parameters θ
Output: ML model that predicts partitioning points given $\bar{\theta}$

Using ML to Accelerate Alpine

Given family of random QCQPs of the form [BST09]

$$\begin{aligned} \nu^*(\theta) := \min_{x,w} \; & c(\theta)^\mathsf{T} x + d(\theta)^\mathsf{T} w \\ \text{s.t.} \; & A(\theta)x + B(\theta)w \leq b, \\ & w_{ij} = x_i x_j, \quad \forall (i,j) \in \mathcal{B}, \\ & x \in [0,1]^{d_x} \end{aligned}$$

Parameters θ vary from one instance to the next

Input: underlying problem, distribution of parameters θ
Output: ML model that predicts partitioning points given $\bar{\theta}$

- Generate N training samples $\{\theta^i\}$ of the problem parameters θ
- Solve max-min problem to determine "optimal" partitioning points for each training instance
- Learn an ML model $\theta^i \mapsto$ `optimal partitioning points`
- Use ML model to predict partitioning points for new instance $\bar{\theta}$

Using ML to Accelerate Alpine

Input: underlying problem, distribution of parameters θ

Output: ML model that predicts partitioning points given $\bar{\theta}$

- Generate 1000 training samples $\{\theta^i\}$ of problem parameters θ
- Solve max-min problem to determine "optimal" partitioning points for each training instance
- Learn an ML model $\theta^i \mapsto$ `optimal partitioning points`
- Use ML model to predict partitioning points for new instance $\bar{\theta}$

Using ML to Accelerate Alpine

Input: underlying problem, distribution of parameters θ

Output: ML model that predicts partitioning points given $\bar{\theta}$

- Generate 1000 training samples $\{\theta^i\}$ of problem parameters θ
- Solve max-min problem to determine "optimal" partitioning points for each training instance
- Learn an ML model $\theta^i \mapsto$ `optimal partitioning points`
- Use ML model to predict partitioning points for new instance $\bar{\theta}$

Use Scikit-learn's AdaBoostRegressor to train Regression Trees with max_depth $= 25$, num_estimators $= 1000$ (no tuning!)

- Features for training and prediction:
  - Parameter θ
  - Best found feasible solution during presolve (one local solve)
  - McCormick lower bounding solution (no partitioning)

Using ML to Accelerate Alpine

Input: underlying problem, distribution of parameters θ

Output: ML model that predicts partitioning points given $\bar{\theta}$

- Generate 1000 training samples $\{\theta^i\}$ of problem parameters θ
- Solve max-min problem to determine "optimal" partitioning points for each training instance
- Learn an ML model $\theta^i \mapsto$ `optimal partitioning points`
- Use ML model to predict partitioning points for new instance $\bar{\theta}$

Use Scikit-learn's AdaBoostRegressor to train Regression Trees with max_depth $= 25$, num_estimators $= 1000$ (no tuning!)

- Features for training and prediction:
  - Parameter θ
  - Best found feasible solution during presolve (one local solve)
  - McCormick lower bounding solution (no partitioning)
- Use 10-fold cross validation to generate predictions for $\{\theta^i\}$

Numerical Experiments on Random QCQPs

Consider random QCQPs of the form [BST09]

$$\begin{aligned} \nu^*(\theta) := \min_{x,w} \; & c(\theta)^\mathsf{T} x + d(\theta)^\mathsf{T} w \\ \text{s.t.} \; & A(\theta)x + B(\theta)w \leq b, \\ & w_{ij} = x_i x_j, \quad \forall (i,j) \in \mathcal{B}, \\ & x \in [0,1]^{d_x} \end{aligned}$$

Parameters θ vary from one instance to the next

Numerical Experiments on Random QCQPs

Consider random QCQPs of the form [BST09]

$$\begin{aligned} \nu^*(\theta) := \min_{x,w} \; & c(\theta)^\mathsf{T} x + d(\theta)^\mathsf{T} w \\ \text{s.t.} \; & A(\theta)x + B(\theta)w \leq b, \\ & w_{ij} = x_i x_j, \quad \forall (i,j) \in \mathcal{B}, \\ & x \in [0,1]^{d_x} \end{aligned}$$

Parameters θ vary from one instance to the next

Consider instances with

- $d_x \in \{10, 20, 50\}$ variables
- $5d_x$ bilinear terms (45 for $d_x = 10$)
- d_x bilinear inequalities
- $d_x/5$ linear equalities

Numerical Results for Random QCQPs

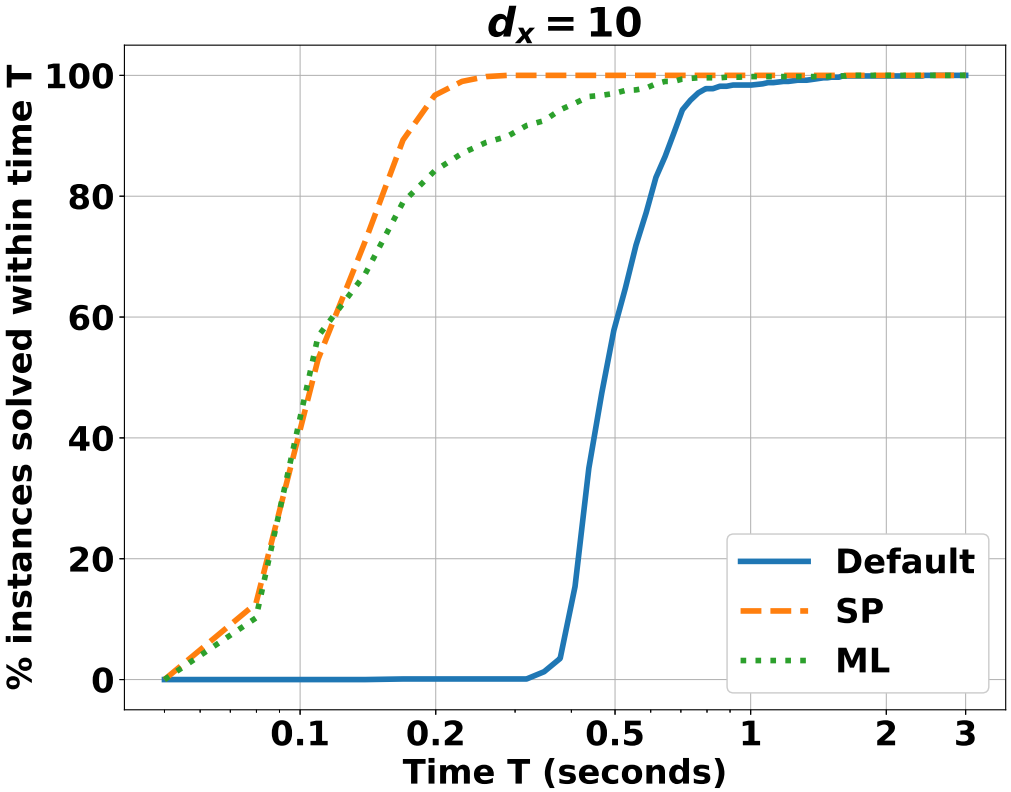
Results for $d_x = 10$ variables

- Generate 1000 random QCQPs with varying parameters θ
- For each instance, determine 2 optimal partitioning points per variable by solving a max-min problem
- Eliminate optimal partitioning points that aren't useful

Numerical Results for Random QCQPs

Results for $d_x = 10$ variables

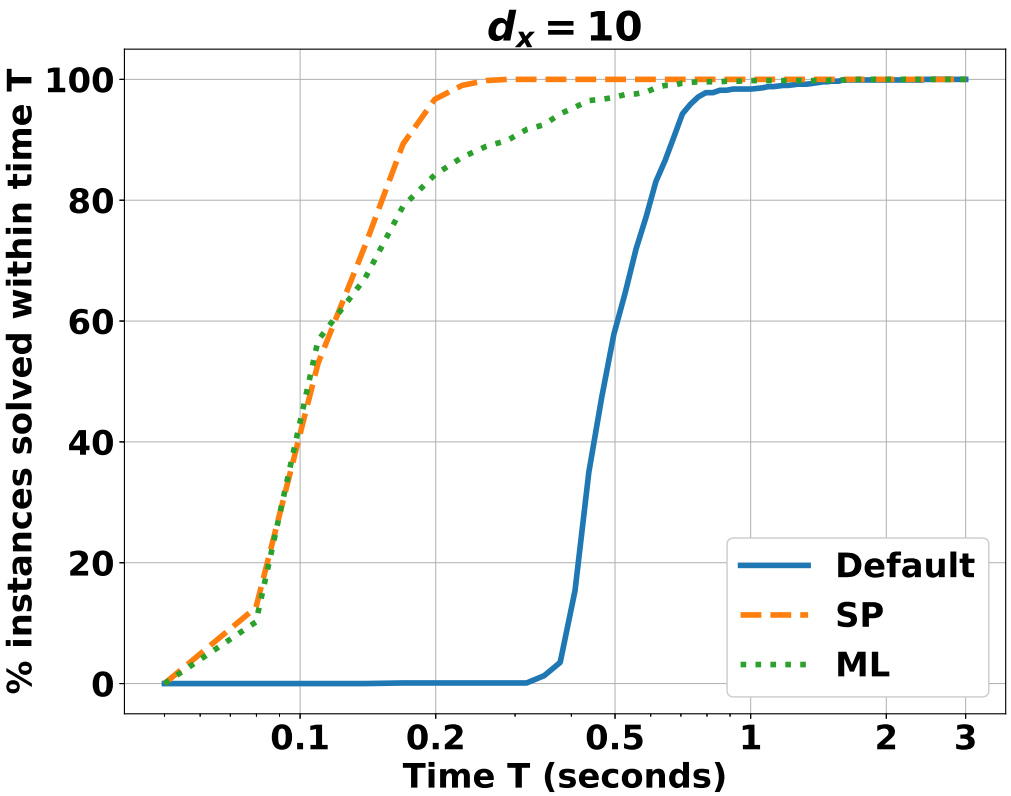
- Generate 1000 random QCQPs with varying parameters θ
- For each instance, determine 2 optimal partitioning points per variable by solving a max-min problem
- Eliminate optimal partitioning points that aren't useful



Numerical Results for Random QCQPs

Results for $d_x = 10$ variables

- Generate 1000 random QCQPs with varying parameters θ
- For each instance, determine 2 optimal partitioning points per variable by solving a max-min problem
- Eliminate optimal partitioning points that aren't useful

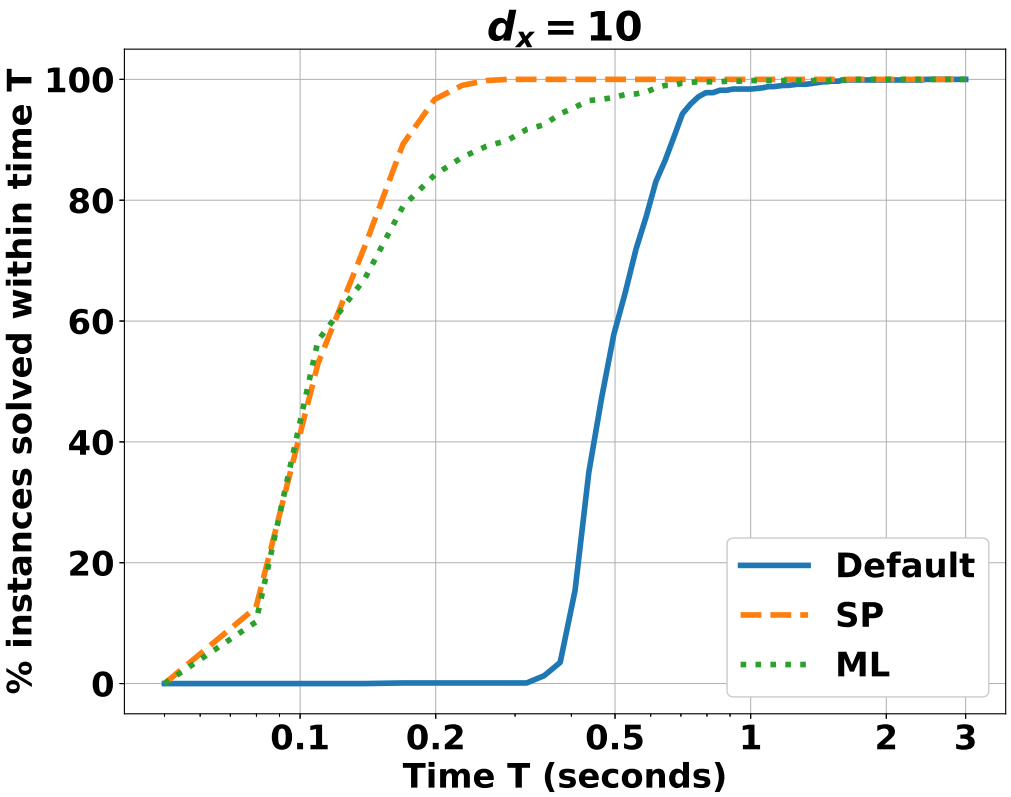


Speedup/ Slowdown	% SP Inst.	% ML Inst.
$1x - 2x$	1.1	7.7
$2x - 3x$	10.2	11.4
$3x - 5x$	47.4	38.5
$5x - 10x$	40.1	40.0
$> 10x$	1.2	0.1
$0.5x - 1x$	—	2.1
$< 0.5x$	—	0.2

Numerical Results for Random QCQPs

Results for $d_x = 10$ variables

- Generate 1000 random QCQPs with varying parameters θ
- For each instance, determine 2 optimal partitioning points per variable by solving a max-min problem
- Eliminate optimal partitioning points that aren't useful



Speedup/ Slowdown	% SP Inst.	% ML Inst.
$1x - 2x$	1.1	7.7
$2x - 3x$	10.2	11.4
$3x - 5x$	47.4	38.5
$5x - 10x$	40.1	40.0
$> 10x$	1.2	0.1
$0.5x - 1x$	—	2.1
$< 0.5x$	—	0.2

Average Speedup (Shifted GM):

Alpine+SP: $4.5x$, Alpine+ML: $3.5x$

Numerical Results for Random QCQPs

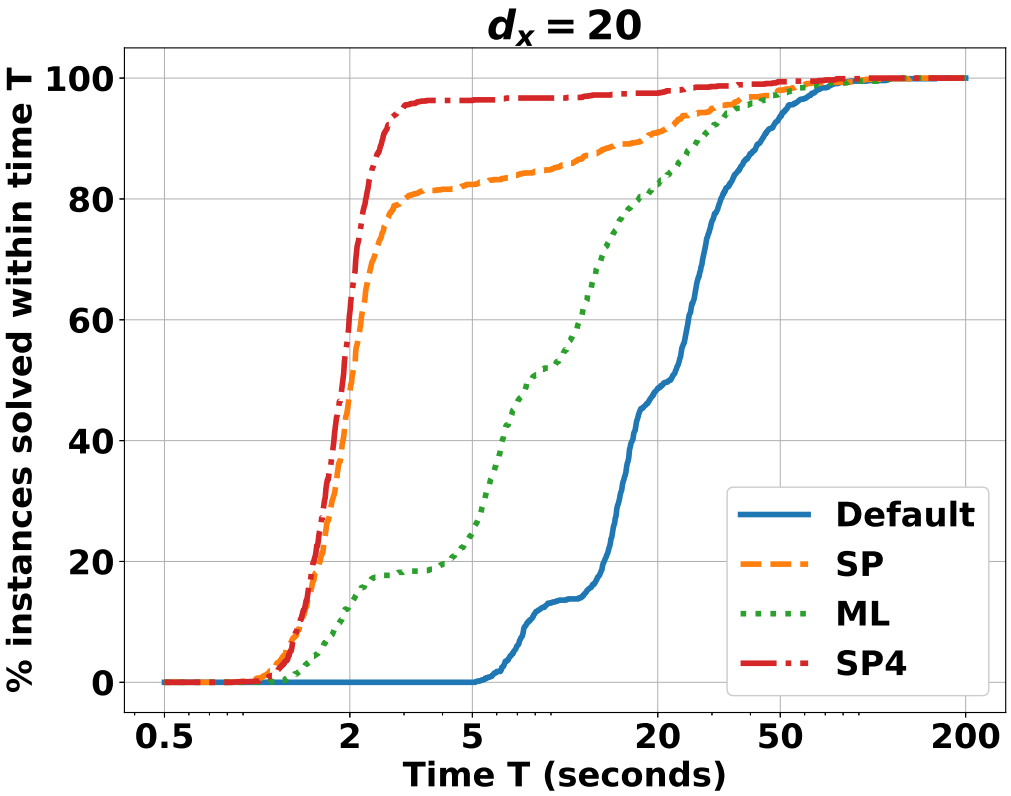
Results for $d_x = 20$ variables

- Generate 1000 random QCQPs with varying parameters θ
- 2/4 partitioning points per variable for each instance
- Eliminate partitioning points that aren't useful

Numerical Results for Random QCQPs

Results for $d_x = 20$ variables

- Generate 1000 random QCQPs with varying parameters θ
- 2/4 partitioning points per variable for each instance
- Eliminate partitioning points that aren't useful



Speedup/ Slowdown	% SP Inst.	% ML Inst.
$1x - 3x$	13.1	48.7
$3x - 5x$	12.3	16.0
$5x - 10x$	31.2	15.3
$10x - 20x$	29.9	6.0
$> 20x$	10.0	0.9
$0.5x - 1x$	3.3	9.8
$< 0.5x$	0.2	3.3

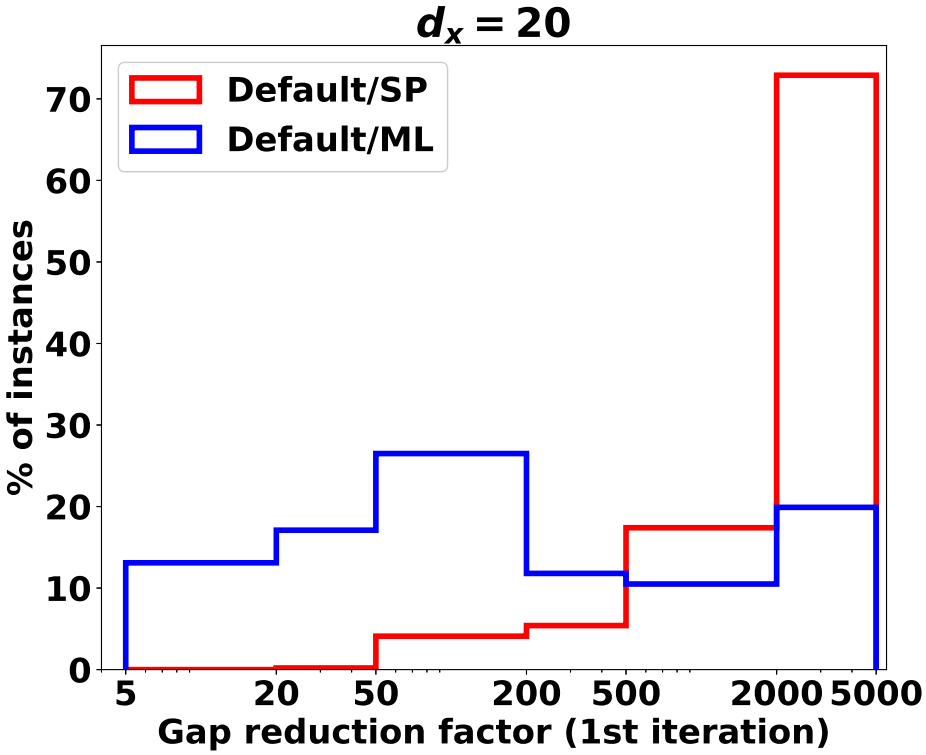
Average Speedup (Shifted GM):

Alpine+SP: $5.1x$, Alpine+ML: $2.1x$

Numerical Results for Random QCQPs

Results for $d_x = 20$ variables

- Generate 1000 random QCQPs with varying parameters θ
- 2/4 partitioning points per variable for each instance
- Eliminate partitioning points that aren't useful



Speedup/ Slowdown	% SP Inst.	% ML Inst.
$1x - 3x$	13.1	48.7
$3x - 5x$	12.3	16.0
$5x - 10x$	31.2	15.3
$10x - 20x$	29.9	6.0
$> 20x$	10.0	0.9
$0.5x - 1x$	3.3	9.8
$< 0.5x$	0.2	3.3

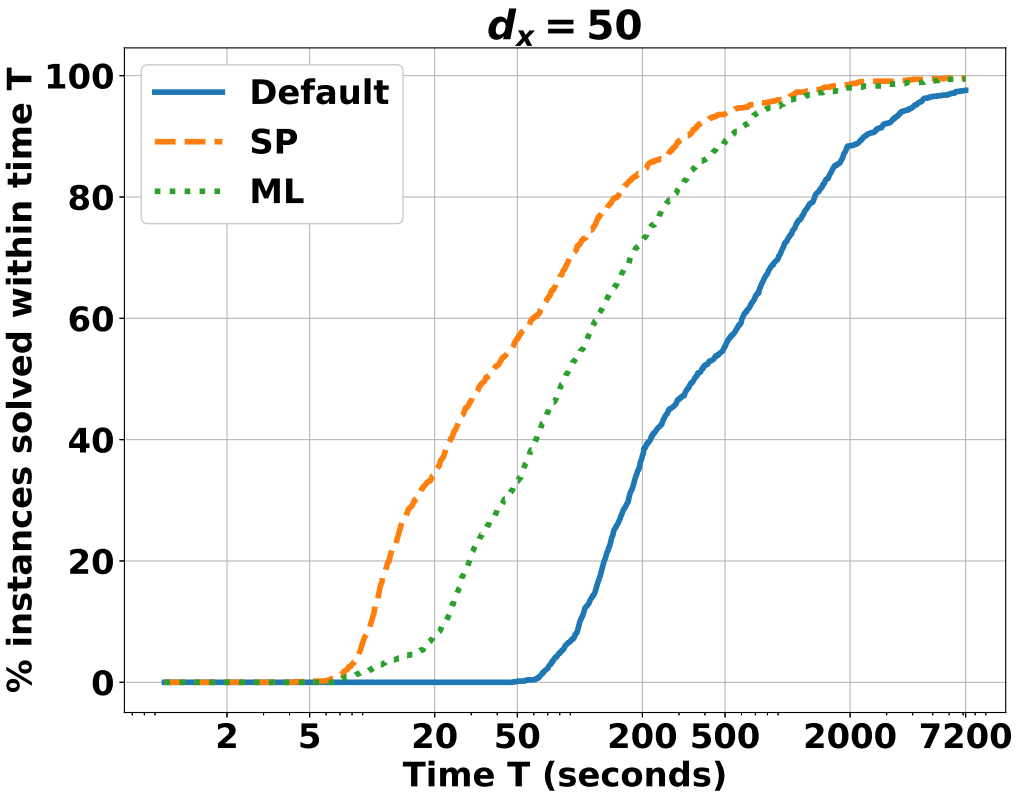
Average Speedup (Shifted GM):

Alpine+SP: $5.1x$, Alpine+ML: $2.1x$

Numerical Results for Random QCQPs

Results for $d_x = 50$ variables

- Generate 1000 random QCQPs with varying parameters θ
- 2 partitioning points per variable for each instance
- Eliminate partitioning points that aren't useful

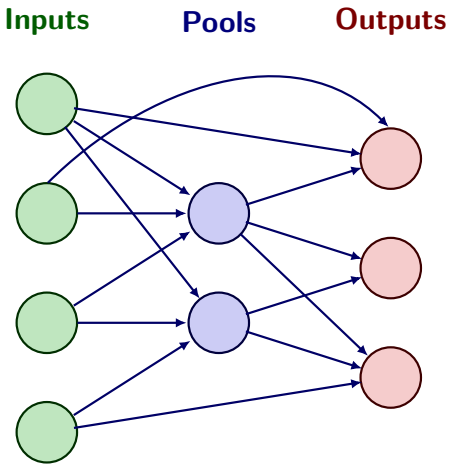


Speedup/ Slowdown	% SP Inst.	% ML Inst.
$1x - 5x$	25.7	49.3
$5x - 10x$	26.3	25.3
$10x - 20x$	24.3	13.7
$20x - 50x$	14.9	5.4
$> 50x$	6.9	0.8
$0.5x - 1x$	1.5	4.8
$< 0.5x$	0.4	0.7

Average Speedup (Shifted GM):

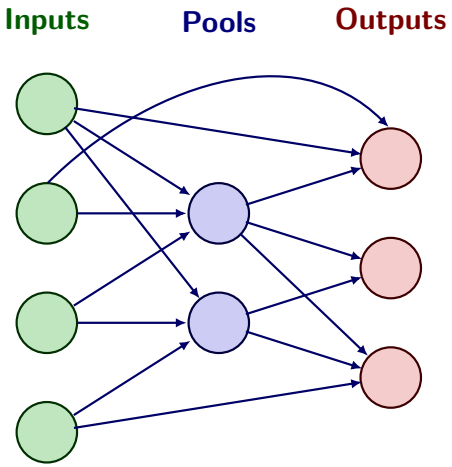
Alpine+SP: $8.1x$, Alpine+ML: $4.2x$

Numerical Results for the Pooling Problem [LdLS20]



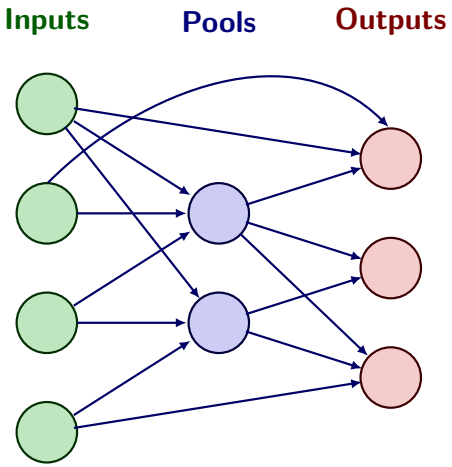
- 45 sources, 15 pools, 30 terminals, 1 quality (124/572 variables part. in 261 bilinear terms)
- 1000 random instances with $\theta =$ input qualities
- 2 partitioning points per variable (total 124×2)

Numerical Results for the Pooling Problem [LdLS20]

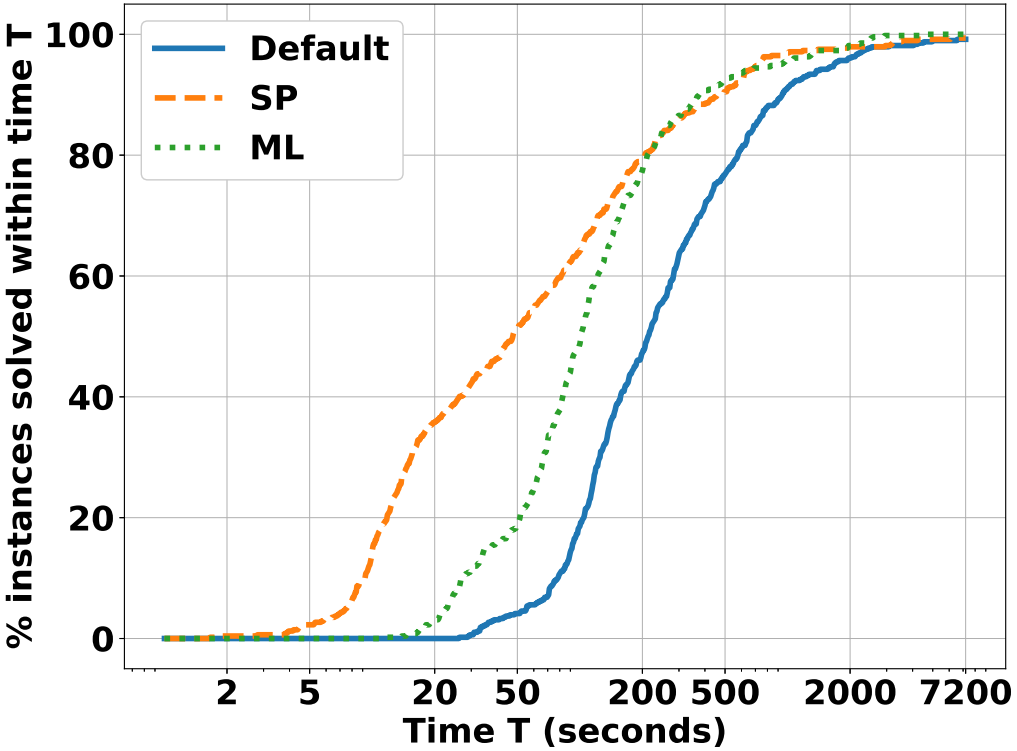


- 45 sources, 15 pools, 30 terminals, 1 quality (124/572 variables part. in 261 bilinear terms)
- 1000 random instances with $\theta =$ input qualities
- 2 partitioning points per variable (total 124×2)
- Feature dimension: 667, Output dimension: 248

Numerical Results for the Pooling Problem [LdLS20]



- 45 sources, 15 pools, 30 terminals, 1 quality (124/572 variables part. in 261 bilinear terms)
- 1000 random instances with θ = input qualities
- 2 partitioning points per variable (total 124×2)
- Feature dimension: 667, Output dimension: 248



Speedup/ Slowdown	% SP Inst.	% ML Inst.
$1x - 3x$	29.1	53.9
$3x - 5x$	16.1	21.5
$5x - 10x$	21.7	10.4
$10x - 20x$	20.3	1.6
$> 20x$	6.2	0.1
$0.5x - 1x$	4.5	1.7
$< 0.5x$	2.1	10.8

Average Speedup (Shifted GM):

Alpine+SP: $3.9x$, Alpine+ML: $2.2x$

Conclusion

- Strong Partitioning can reduce Alpine's solution time by $4x - 9x$ on average
- Strong Partitioning can reduce Alpine's first iteration gap by more than three orders of magnitude!
- Off-the-shelf ML model can improve Alpine's run time by $2x - 4.5x$ on average

Conclusion

- Strong Partitioning can reduce Alpine's solution time by $4x - 9x$ on average
- Strong Partitioning can reduce Alpine's first iteration gap by more than three orders of magnitude!
- Off-the-shelf ML model can improve Alpine's run time by $2x - 4.5x$ on average

Future Work:

- Techniques for sparse partitioning
- Train more advanced ML models
- Extension to broader optimization classes, including mixed-integer problems
- Explore application to AC-OPF

Questions? rohitk@alum.mit.edu

References I

[ALW17] Alejandro Marcos Alvarez, Quentin Louveaux, and Louis Wehenkel. A machine learning-based approximation of strong branching. *INFORMS Journal on Computing*, 29(1):185–195, 2017.

[BLBMT19] Radu Baltean-Lugojan, Pierre Bonami, Ruth Misener, and Andrea Tramontani. Scoring positive semidefinite cutting planes for quadratic optimization via trained neural networks. *preprint: http://www. optimization-online. org/DB_ HTML/2018/11/6943. html*, 2019.

[BLZ18] Pierre Bonami, Andrea Lodi, and Giulia 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.

[BST09] Xiaowei Bao, Nikolaos V Sahinidis, and Mohit Tawarmalani. Multiterm polyhedral relaxations for nonconvex, quadratically constrained quadratic programs. *Optimization Methods & Software*, 24(4-5):485–504, 2009.

[CNB+22] Fatih Cengil, Harsha Nagarajan, Russell Bent, Sandra Eksioglu, and Burak Eksioglu. Learning to accelerate globally optimal solutions to the AC Optimal Power Flow problem. *Electric Power Systems Research*, 212:108275, 2022.

[GCF+19] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *arXiv preprint arXiv:1906.01629*, 2019.

[GGCGD+22] Bissan Ghaddar, Ignacio Gómez-Casares, Julio González-Díaz, Brais González-Rodríguez, Beatriz Pateiro-López, and Sofía Rodríguez-Ballesteros. Learning for spatial branching: An algorithm selection approach. *arXiv preprint arXiv:2204.10834*, 2022.

[GRAPAP+22] Brais González-Rodríguez, Raúl Alvite-Pazó, Samuel Alvite-Pazó, Bissan Ghaddar, and Julio González-Díaz. Polynomial optimization: Enhancing RLT relaxations with conic constraints. *arXiv preprint arXiv:2208.05608*, 2022.

[KLBS+16] Elias Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[LdLS20] James Luedtke, Claudia d'Ambrosio, Jeff Linderoth, and Jonas Schweiger. Strong convex nonlinear relaxations of the pooling problem. *SIAM Journal on Optimization*, 30(2):1582–1609, 2020.

References II

[NBG+20] Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020.

[NBL+11] Giacomo Nannicini, Pietro Belotti, Jon Lee, Jeff Linderoth, François Margot, and Andreas Wächter. A probing algorithm for MINLP with failure prediction by SVM. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 154–169. Springer, 2011.

[NLW+19] Harsha Nagarajan, Mowen Lu, Site Wang, Russell Bent, and Kaarthik Sundar. An adaptive, multivariate partitioning algorithm for global optimization of nonconvex programs. *Journal of Global Optimization*, 74(4):639–675, 2019.