Mixed-Integer Convex Nonlinear Optimization with Gradient-Boosted Trees Embedded

Miten Mistry

Dimitrios Letsios
Imperial College London, South Kensington, SW7 2AZ, UK.

Gerhard Krennrich

Robert M. Lee
BASF SE, Ludwigshafen am Rhein, Germany.

Ruth Misener
Imperial College London, South Kensington, SW7 2AZ, UK.

Decision trees usefully represent sparse, high dimensional and noisy data. Having learned a function from this data, we may want to thereafter integrate the function into a larger decision-making problem, e.g., for picking the best chemical process catalyst. We study a large-scale, industrially-relevant mixed-integer nonlinear nonconvex optimization problem involving both gradient-boosted trees and penalty functions mitigating risk. This mixed-integer optimization problem with convex penalty terms broadly applies to optimizing pre-trained regression tree models. Decision makers may wish to optimize discrete models to repurpose legacy predictive models, or they may wish to optimize a discrete model that particularly well-represents a data set. We develop several heuristic methods to find feasible solutions, and an exact, branch-and-bound algorithm leveraging structural properties of the gradient-boosted trees and penalty functions. We computationally test our methods on concrete mixture design instance and a chemical catalysis industrial instance.

Key words: Gradient-boosted trees, branch-and-bound

1. Introduction

Consider integrating an unknown function into an optimization problem, i.e., without a closed-form formula, but with a data set representing evaluations over a box-constrained feasibility domain. Optimization in the machine learning literature usually refers to the training procedure, e.g., model accuracy maximization (Sra et al. 2012, Snoek et al. 2012). This paper investigates optimization problems after the training procedure, where the trained predictive model is embedded in the optimization problem. We consider optimization methods for problems with gradient-boosted tree (GBT) models embedded (Friedman 2001, Hastie et al. 2009). Advantages of GBTs are myriad (Chen and Guestrin 2016, Ke et al. 2017), e.g., they are robust to scale differences in the training data features, handle both

categorical and numerical variables, and can minimize arbitrary, differentiable loss functions. GBTs are interpretable models, i.e., humans can understand their enclosed information.

After developing a machine learning model, we may wish to embed the model into a larger decision-making problem. The resulting optimization models may be addressed using local (Nocedal and Wright 2006) or deterministic global (Schweidtmann and Mitsos 2018) methods. The value of global optimization is known in engineering (Boukouvala et al. 2016), e.g., local minima can lead to infeasible parameter estimation (Singer et al. 2006) or misinterpreted data (Bollas et al. 2009). For applications where global optimization is less relevant, we still wish to develop optimization methods for discrete and non-smooth machine learning models, e.g., regression trees. Discrete optimization methods allow repurposing a legacy model, originally built for prediction, into an optimization framework.

Our optimization problem incorporates an additional convex penalty term in the objective, accounting for risky predicted values in parts of the feasibility domain where the machine learning model is not well trained due to missing data. For instance, consider historical data from a manufacturing process for quality maximization. The data may exhibit correlation between two process parameters, e.g., the temperature and the concentration of a chemical additive. A machine learned model of the system assigns weights to these parameters for future predictions. Lacking additional information, numerical optimization may produce candidate solutions with temperature and concentration combinations that (possibly incorrectly) suggest temperature is responsible for an observed effect. The convex penalty term helps control the optimizer's adventurousness by penalizing deviation from the training data subspace and is parameterized using principal component analysis (Vaswani et al. 2018). Large values of this risk control term generate conservative solutions. Smaller penalty values explore regions with greater possible rewards but also additional risk.

This paper considers a mixed-integer nonlinear optimization problem with convex nonlinearities (convex MINLP). The objective sums a discrete GBT-trained function and a continuous convex penalty function. We design exact methods computing either globally optimal solutions, or solutions within a quantified distance from the global optimum. The convex MINLP formulation enables us to solve industrial instances with commercial solvers. We develop a new branch-and-bound method exploiting both the GBTs combinatorial structure and the penalty function convexity. Numerical results substantiate our approach.

Paper organization Section 3 introduces the optimization problem and Section 4 formulates it as a convex MINLP. Section 5 describes our branch-and-bound method. Section 6 defines the convex penalty term. Section 7 presents numerical results. Finally, Section 8 concludes.

2. Background

This section describes gradient-boosted trees (GBTs) (Friedman 2001, 2002). In this work, GBTs are embedded into the Section 3 optimization problem. GBTs are a subclass of boosting methods (Freund 1995). Boosting methods train several weak learners iteratively that collectively produce a strong learner, where a weak learner is at least better than random guessing. Each boosting iteration trains a new weak learner against the residual of the previously trained learners by minimizing a loss function. For GBTs, the weak learners are classification and regression trees (Breiman et al. 1984).

This paper restricts its analysis to regression GBTs without categorical input variables. A trained GBT function is a collection of binary trees and each of these trees provides its own independent contribution when evaluating at $\boldsymbol{x}$.

DEFINITION 1. A trained GBT function is defined by sets $(\mathcal{T}, \mathcal{V}_t, \mathcal{L}_t)$ and values $(i(t,s), v(t,s), F_{t,l})$. Set $\mathcal{T}$ indexes the trees. For a given tree $t \in \mathcal{T}$, $\mathcal{V}_t$ and $\mathcal{L}_t$ index the split and leaf nodes, respectively. At split node $t \in \mathcal{T}$, $s \in \mathcal{V}_t$, $i(t,s)$ and $v(t,s)$ return the split variable and value, respectively. At leaf node $t \in \mathcal{T}$, $l \in \mathcal{L}_t$, $F_{t,l}$ is its contribution.

Tree $t \in \mathcal{T}$ evaluates at $\boldsymbol{x}$ by following a root-to-leaf path. Beginning at the root node of t, each encountered split node $s \in \mathcal{V}_t$ assesses whether $x_{i(t,s)} < v(t,s)$ or $x_{i(t,s)} \geq v(t,s)$ and follows the left or right child, respectively. The leaf $l \in \mathcal{L}_t$ corresponding to $\boldsymbol{x}$ returns t's contribution $F_{t,l}$. Figure 1 shows how a single gradient-boosted tree recursively partitions the domain. The overall output, illustrated in Figure 2, sums all individual tree evaluations:

$$\mathrm{GBT}(\boldsymbol{x}) = \sum_{t \in \mathcal{T}} \mathrm{GBT}_t(\boldsymbol{x}).$$

EXAMPLE 1. Consider a trained GBT that approximates a two-dimensional function with $\mathcal{T} = \{t_1, \ldots, t_{|\mathcal{T}|}\}$. To evaluate $\mathrm{GBT}(\boldsymbol{x})$ where $\boldsymbol{x} = (4.2, 2.8)^\top$, let t_1 be the tree given by Figure 1, the highlighted path corresponds to evaluating at $\boldsymbol{x}$. The root split node query of $x_1 < 2$ is false, since $x_1 = 4.2$, so we follow the right branch. Following this branch encounters another split node. The next query of $x_2 < 4$ is true, since $x_2 = 2.8$, so we follow the left branch. The final branch reaches a leaf with value 4.3, hence $\mathrm{GBT}_{t_1}(\boldsymbol{x}) = 4.3$. The remaining trees also return a value after making similar queries on $\boldsymbol{x}$. This results in $\mathrm{GBT}(\boldsymbol{x}) = \sum_{i=1}^{|\mathcal{T}|} \mathrm{GBT}_{t_i}(\boldsymbol{x}) = 4.3 + \sum_{i=2}^{|\mathcal{T}|} \mathrm{GBT}_{t_i}(\boldsymbol{x})$.

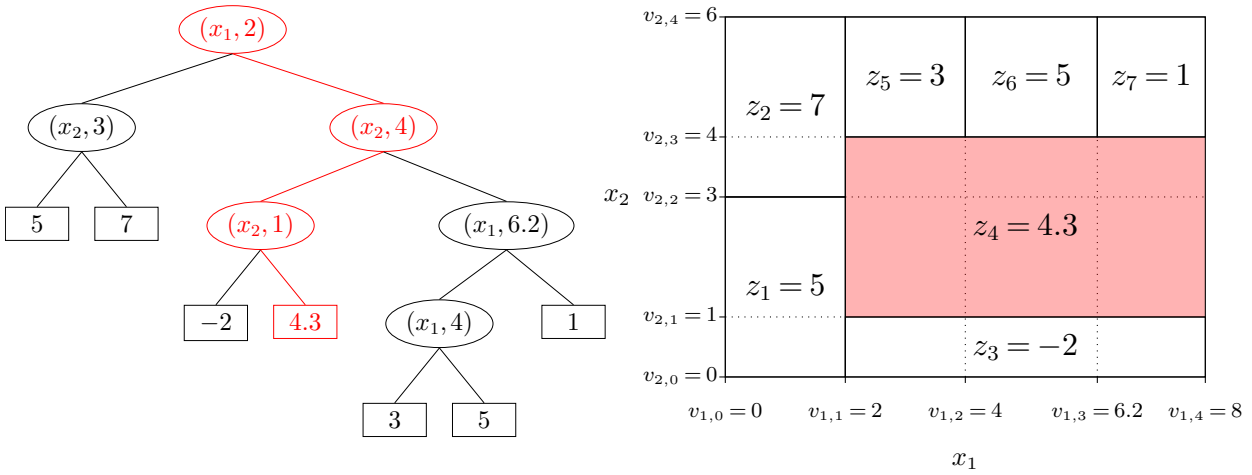

Figure 1 **Gradient boosted tree, see Definition 1, trained in two dimensions. Left: gradient boosted tree. Right: recursive domain partition defined by tree on left. The highlighted path and region corresponds to the result of evaluating at $\boldsymbol{x} = (4.2, 2.8)^\top$ as in Example (1).**

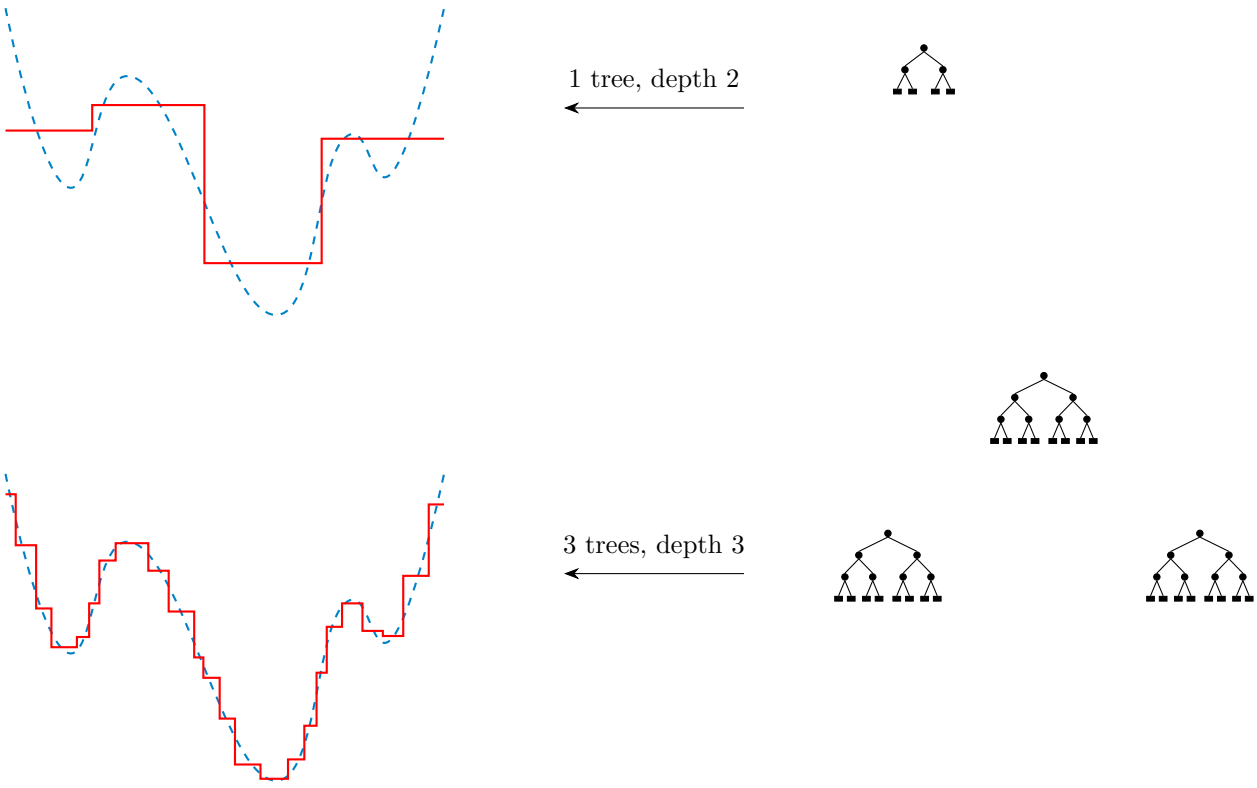

Figure 2 **GBT approximations to the dashed function: 1 tree of depth 2 (top) and 3 trees of depth 3 (bottom).**

3. Optimization Problem

This paper considers box-constrained optimization Problem (1) summing two objective components: a convex nonlinear function and a GBT-trained function. Problem (1) is

Table 1 **Mixed-integer convex programming model sets, parameters and variables.**

Symbol	Description
v_i^L, v_i^U	Lower and upper bound of variable x_i
x_i	Continuous variable, $i \in \{1, \ldots, n\}$
$t \in \mathcal{T}$	Indices of GBTs
$l \in \mathcal{L}_t$	Indices of leaves for tree t
$s \in \mathcal{V}_t$	Indices of split nodes for tree t
m_i	Number of variable x_i splitting values
$v_{i,j}$	Variable i's j-th breakpoint, $j \in \{1, \ldots, m_i\}$
$F_{t,l}$	Value of leaf (t, l)
$y_{i,j}$	Binary variable indicating whether variable $x_i < v_{i,j}$
$z_{t,l}$	Nonnegative variable that activates leaf (t, l)

relevant, for example, in cases where a GBT function has been trained to data but we may trust an optimal solution close to regions with many training points. A convex penalty term may penalize solutions further from training data:

$$\min_{\boldsymbol{v}^L \le \boldsymbol{x} \le \boldsymbol{v}^U} \underbrace{\text{cvx}(\boldsymbol{x})}_{\textit{Convex Part}} + \underbrace{\text{GBT}(\boldsymbol{x})}_{\textit{GBT Part}}, \tag{1}$$

where $\boldsymbol{x} = (x_1, \ldots, x_n)^\top$ is the variable vector. $\text{GBT}(\boldsymbol{x})$ is the GBT-trained function value at $\boldsymbol{x}$. Table 1 defines the model sets, parameters and variables.

A given problem instance may sum independently-trained GBT functions. W.l.o.g., we equivalently optimize a single GBT function which is the union of all original GBTs.

4. Mixed-Integer Convex Formulation

Problem (1) consists of a continuous convex function and a discrete GBT function. The discrete nature of the GBT function arises from the left/right decisions at the split nodes. So we consider a mixed-integer nonlinear program with convex nonlinearities (convex MINLP) formulation. The main ingredient of the convex MINLP model is a mixed-integer linear programming (MILP) formulation of the GBT part which merges with the convex part via a linking constraint. The high level convex MINLP is:

$$\min_{\boldsymbol{v}^L \le \boldsymbol{x} \le \boldsymbol{v}^U} \quad \text{cvx}(\boldsymbol{x}) + [\text{GBT MILP objective}] \tag{2a}$$

$$\text{s.t.} \quad [\text{GBT MILP constraints}], \tag{2b}$$

$$[\text{Variable linking constraints}]. \tag{2c}$$

MILP approaches for machine learning show competitive performance for important applications (Bertsimas and Mazumder 2014, Miyashiro and Takano 2015, Bertsimas and

King 2016, Bertsimas et al. 2016, Günlük et al. 2016, Bertsimas and Dunn 2017, Fischetti and Jo 2018). The MILP method performance arises from strong mixed-integer algorithms and solver availability (Bixby 2012).

4.1. GBT MILP Formulation

We use a MILP formulation to leverage commercial MILP codes, alternatives to MILP include constraint programming (Rossi et al. 2006) and satisfiability modulo theories (Mistry et al. 2018). We form the GBT MILP using the Mišić (2017) approach. Figure 1 shows how a GBT partitions the domain $[\boldsymbol{v}^L, \boldsymbol{v}^U]$ of $\boldsymbol{x}$. Optimizing a GBT function reduces to optimizing the leaf selection, i.e., finding an optimal interval, opposed to a specific $\boldsymbol{x}$ value. Aggregating over all GBT split nodes produces a vector of ordered breakpoints $v_{i,j}$ for each x_i variable: $v_i^L = v_{i,0} < v_{i,1} < \cdots < v_{i,m_i} < v_{i,m_i+1} = v_i^U$. Selecting a consecutive pair of breakpoints for each x_i defines an interval where the GBT function is constant. Each point $x_i \in [v_i^L, v_i^U]$ is either on a breakpoint $v_{i,j}$ or in the interior of an interval. Binary variable $y_{i,j}$ models whether $x_i < v_{i,j}$ for $i \in [n] = \{1, \ldots, n\}$ and $j \in [m_i] = \{1, \ldots, m_i\}$. Binary variable $z_{t,l}$ is 1 if tree $t \in \mathcal{T}$ evaluates at node $l \in \mathcal{L}_t$ and 0 otherwise. Denote by $\mathcal{V}_t$ the set of split nodes for tree t. Moreover, let $\text{Left}_{t,s}$ and $\text{Right}_{t,s}$ be the sets of subtree leaf nodes rooted in the left and right children of split node s, respectively.

MILP Problem (3) formulates the GBT (Mišić 2017). Equation (3a) minimizes the total value of the active leaves. Equation (3b) selects exactly one leaf per tree. Equations (3c) and (3d) activates a leaf only if all corresponding splits occur. Equation (3e) ensures that if $x_i \leq v_{i,j-1}$, then $x_i \leq v_{i,j}$. W.l.o.g, we drop the $z_{t,l}$ integrality constraint because any feasible assignment of $\boldsymbol{y}$ specifies one leaf, i.e., a single region in Figure 1. The MILP formulation recalls the state-of-the-art in modeling piecewise linear functions (Misener et al. 2009, Misener and Floudas 2010, Vielma et al. 2010).

$$\min \sum_{t \in \mathcal{T}} \sum_{l \in \mathcal{L}_t} F_{t,l} z_{t,l} \tag{3a}$$

$$\text{s.t.} \sum_{l \in \mathcal{L}_t} z_{t,l} = 1, \qquad \forall t \in \mathcal{T}, \tag{3b}$$

$$\sum_{l \in \text{Left}_{t,s}} z_{t,l} \leq y_{i(s),j(s)}, \qquad \forall t \in \mathcal{T}, s \in \mathcal{V}_t, \tag{3c}$$

$$\sum_{l \in \text{Right}_{t,s}} z_{t,l} \leq 1 - y_{i(s),j(s)}, \qquad \forall t \in \mathcal{T}, s \in \mathcal{V}_t, \tag{3d}$$

$$y_{i,j} \leq y_{i,j+1}, \qquad \forall i \in [n],\, j \in [m_i - 1], \tag{3e}$$

$$y_{i,j} \in \{0,1\}, \qquad \forall i \in [n],\, j \in [m_i], \tag{3f}$$

$$z_{t,l} \geq 0, \qquad \forall t \in \mathcal{T},\, l \in \mathcal{L}_t. \tag{3g}$$

4.2. Linking Constraints

Equations (4a) and (4b) relate the continuous x_i variables, from the original Problem (1) definition, to the binary $y_{i,j}$ variables:

$$x_i \geq v_{i,0} + \sum_{j=1}^{m_i} (v_{i,j} - v_{i,j-1})(1 - y_{i,j}), \tag{4a}$$

$$x_i \leq v_{i,m_i+1} + \sum_{j=1}^{m_i} (v_{i,j} - v_{i,j+1}) y_{i,j}, \tag{4b}$$

for all $i \in [n]$. We express the linking constraints using non-strict inequalities to avoid computational issues when optimizing with strict inequalities. Combining Equations (2) to (4) defines the mixed-integer nonlinear program with convex nonlinearities (convex MINLP) formulation to Problem (1). Appendix A lists the complete formulation.

4.3. Worst Case Analysis

The difficulty of Problem (1) is primarily justified by the fact that optimizing a GBT-trained function, i.e., Problem (3), is an NP-hard problem (Mišić 2017). This section shows that the number of continuous variable splits and tree depth affect complete enumeration methods. These parameters motivate the branching scheme in our branch-and-bound algorithm.

In a GBT ensemble, each continuous variable x_i is associated with $m_i + 1$ intervals (splits). Picking one interval $j \in \{1, \ldots, m_i + 1\}$ for each x_i sums to a total of $\prod_{i=1}^{n}(m_i + 1)$ distinct combinations. A GBT-trained function evaluation selects a leaf from each tree. But not all leaf combinations are valid evaluations. In a feasible leaf combination where one leaf enforces $x_i < v_1$ and another enforces $x_i \geq v_2$, it must be that $v_2 < v_1$. Let d be the maximum tree depth in $\mathcal{T}$. Then, the number of leaf combinations is upper bounded by $2^{d|\mathcal{T}|}$. Since the number of feasibility checks for a single combination is $\frac{1}{2}|\mathcal{T}|(|\mathcal{T}|-1)$, an upper bound on the total number of feasibility checks is $2^{d|\mathcal{T}|-1}|\mathcal{T}|(|\mathcal{T}|-1)$. This observation implies that the worst case performance of an exact method improves as the number of trees decreases.

5. Branch-and-Bound Algorithm

This section designs an exact branch-and-bound (B&B) approach. Using a divide-and-conquer principle, B&B forms a tree of subproblems and searches the domain of feasible solutions. Key aspects of B&B are: (i) rigorous lower (upper) bounding methods for minimization (maximization) subproblems, (ii) branch variable and value selection, and (iii) feasible solution generation. In the worst case, B&B enumerates all solutions, but generally it avoids complete enumeration by pruning subproblems, i.e., removing infeasible subproblems or nodes with lower bound exceeding the best found feasible solution (Morrison et al. 2016). This section exploits *spatial branching* that splits on continuous variables (Belotti et al. 2013). Table 5 in Appendix E defines the symbols in this section.

5.1. Overview

B&B Algorithm 1 spatially branches over the $[\boldsymbol{v}^L, \boldsymbol{v}^U]$ domain. It selects a variable x_i, a point v and splits interval $[v_i^L, v_i^U]$ into intervals $[v_i^L, v]$ and $[v, v_i^U]$. Each interval corresponds to an independent subproblem and a new B&B node. To avoid redundant branches, all GBT splits define the B&B branching points. At a given node, denote the reduced node domain by $S = [\boldsymbol{L}, \boldsymbol{U}]$. Algorithm 1 solves Problem (1) by relaxing the Equation (4) linking constraints and thereby separating the convex and GBT parts. Using this separation, Algorithm 1 computes corresponding bounds $b^{\text{cvx},S}$ and $b^{\text{GBT},S,P}$ independently, where the latter bound requires a tree ensemble partition P initialized at the root node and dynamically refined at each non-root node.

Algorithm 1 begins by constructing the root node, computing a global lower bound, and determining a global ordering of all branches (lines 1–5). A given iteration: (i) extracts a node S from the unexplored node set Q, (ii) strong branches at S to cheaply identify branches that tighten the domain resulting in node S', (iii) updates the GBT lower bound at S', (iv) branches to obtain the child nodes S_{left} and S_{right}, (v) assesses if each child node $S_{\text{child}} \in \{S_{\text{left}}, S_{\text{right}}\}$ may now be pruned and, if not, (vi) adds S_{child} to the unexplored node set Q (lines 8–25).

The remainder of this section is structured as follows. Section 5.2 lower bounds Problem (1). Section 5.3 introduces a GBT branch ordering and leverages strong branching for cheap node pruning. Section 5.4 discusses heuristics for computing efficient upper bounds.

Algorithm 1 Branch-and-Bound (B&B) Algorithm Overview

$S = [\boldsymbol{L}, \boldsymbol{U}] \leftarrow [\boldsymbol{v}^L, \boldsymbol{v}^U]$
$b^{\text{cvx},S} \leftarrow \textsc{ConvexBound}(S)$ ▷ Lemma 1, Section 5.2.1
$P_{\text{root}} \leftarrow \textsc{RootNodePartition}(N)$ ▷ Section 5.2.2
$b^{\text{GBT},S,P_{\text{root}}} \leftarrow \textsc{GbtBound}(S, P_{\text{root}})$ ▷ Lemma 2, Section 5.2.2
$B \leftarrow \textsc{BranchOrdering}()$ ▷ Section 5.3.1
$Q = \{S\}$
while $Q \neq \emptyset$ **do**
    Select $S \in Q$
    if S is not leaf **then**
        $S' \leftarrow S$
        repeat
            $S', (x_i, v) \leftarrow \textsc{StrongBranch}(S', B)$ ▷ Algorithm 3, Section 5.3.2
        until strong branch not found
        if S' is not leaf **then**
            $(S_{\text{left}}, S_{\text{right}}) \leftarrow \textsc{Branch}(S', (x_i, v))$
            P: tree ensemble partition of node S
            $P' \leftarrow \textsc{PartitionRefinement}(P)$ ▷ Algorithm 2, Section 5.2.2
            $b^{\text{GBT},S',P'} \leftarrow \textsc{GbtBound}(S', P')$ ▷ Lemma 2, Section 5.2.2
            for $S_{\text{child}} \in \{S_{\text{left}}, S_{\text{right}}\}$ **do**
                if S_{child} cannot be pruned **then** ▷ Section 5.2.3
                    $Q \leftarrow Q \cup \{S_{\text{child}}\}$
                end if
            end for
        end if
    end if
    $Q \leftarrow Q \setminus \{S\}$
end while

5.2. Lower Bounding

5.2.1. Global lower bound The convex MINLP Problem (2) objective function consists of a *convex* (penalty) part and a *mixed-integer linear* (GBT) part. Lemma 1 computes a lower bound on the problem by handling the convex and GBT parts independently.

LEMMA 1. *Let $S = [\boldsymbol{L}, \boldsymbol{U}] \subseteq [\boldsymbol{v}^L, \boldsymbol{v}^U]$ be a sub-domain of optimization Problem (2). Denote by R^S the optimal objective value, i.e., the tightest relaxation, over the sub-domain S. Then, it holds that $R^S \geq \hat{R}^S$, where:*

$$\hat{R}^S = \underbrace{\left[\min_{\boldsymbol{x} \in S} \ \mathrm{cvx}(\boldsymbol{x})\right]}_{b^{\mathrm{cvx},S}} + \underbrace{\left[\min_{\boldsymbol{x} \in S} \sum_{t \in \mathcal{T}} \mathrm{GBT}_t(\boldsymbol{x})\right]}_{b^{\mathrm{GBT},S,*}}.$$

Proof Let $\boldsymbol{x}^* = \arg\min_{\boldsymbol{x} \in S}\{\mathrm{cvx}(\boldsymbol{x}) + \mathrm{GBT}(\boldsymbol{x})\}$ and observe that $\mathrm{cvx}(\boldsymbol{x}^*) \geq b^{\mathrm{cvx},S}$ and $\mathrm{GBT}(\boldsymbol{x}^*) \geq b^{\mathrm{GBT},S,*}$. □

Lemma 1 treats the two Problem (1) objective terms independently, i.e., $\hat{R}^S$ separates the convex part from the GBT part. We may compute $\hat{R}^S$ by removing the Equation (4) linking constraints and solving the mixed-integer model consisting of Equations (2) and (3). Computationally, the Lemma 1 separation leverages efficient algorithms for the convex part and commercial codes for the MILP GBT part.

5.2.2. GBT Lower Bound While we may efficiently compute $b^{\mathrm{cvx},S}$ (Boyd and Vandenberghe 2004), deriving $b^{\mathrm{GBT},S,*}$ is $\mathcal{NP}$-hard (Mišić 2017). With the aim of tractability, we calculate a relaxation of $b^{\mathrm{GBT},S,*}$. Lemma 2 lower bounds Problem (3), i.e., the GBT part of Problem (2), by partitioning the GBT ensemble into a collection of smaller ensembles.

LEMMA 2. *Consider a sub-domain $S = [\boldsymbol{L}, \boldsymbol{U}] \subseteq [\boldsymbol{v}^L, \boldsymbol{v}^U]$ of the optimization problem. Let $P = \{\mathcal{T}_1, \ldots, \mathcal{T}_k\}$ be any partition of $\mathcal{T}$, i.e., $\cup_{i=1}^k \mathcal{T}_i = \mathcal{T}$ and $\mathcal{T}_i \cap \mathcal{T}_j = \emptyset \ \forall 1 \leq i < j \leq k$. Then, it holds that $b^{\mathrm{GBT},S,*} \geq b^{\mathrm{GBT},S,P}$, where:*

$$b^{\mathrm{GBT},S,P} = \sum_{\mathcal{T}' \in P} \left[\min_{\boldsymbol{x} \in S} \left\{\sum_{t \in \mathcal{T}'} \mathrm{GBT}_t(\boldsymbol{x})\right\}\right].$$

Proof When evaluating $\mathrm{GBT}(\boldsymbol{x})$ at a given $\boldsymbol{x}$, each tree $t \in \mathcal{T}$ provides its own independent contribution $\mathrm{GBT}_t(\boldsymbol{x})$, i.e., a single leaf. A feasible selection of leaves has to be consistent with respect to the GBT node splits, i.e., if one leaf splits on $x_i < v_1$ and another splits on $x_i \geq v_2$ then $v_1 > v_2$. Relaxing this consistency requirement by considering a partition P of $\mathcal{T}$ derives the lower bounds $b^{\mathrm{GBT},S,P}$ for any partition P. □

Root Node Partition B&B Algorithm 1 chooses an initial root node partition P_{root} with subsets of size N and calculates the associated Lemma 2 lower bound. Section 7 numerically decides the partition size N for the considered instances. The important factors for a subset size N are the tree depth, the number of continuous variable splits and their relation with the number of binary variables.

Non-Root Node Partition Refinement Any non-root B&B node has reduced domain $\boldsymbol{x} \in S = [\boldsymbol{L}, \boldsymbol{U}] \subset [\boldsymbol{v}^L, \boldsymbol{v}^U]$. B&B Algorithm 1 only branches on GBT node splits, so modeling the reduced domain S in MILP Problem (3) is equivalent to setting $y_{i,j} = 0$ or $y_{i,j} = 1$ for any $y_{i,j}$ that corresponds to $x_i \leq L_i$ or $x_i \geq U_i$, respectively.

Assume that, at some non-root node with domain S, the algorithm is about to update $b^{\text{GBT},S',P'}$ which was calculated at the parent node with domain $S' \supset S$. Fixing binary variables $y_{i,j}$ subject to domain S reduces the worst case enumeration cost of calculating $b^{\text{GBT},S,P'}$. The GBT lower bound may further improve at S by considering an alternative partition P such that $|P| < |P'|$, i.e., reducing the number of subsets. However, reducing the number of subsets has challenges because: (i) choosing any partition P does not necessarily guarantee $b^{\text{GBT},S,P} \geq b^{\text{GBT},S',P'}$, and (ii) a full Lemma 2 calculation of $b^{\text{GBT},S,P}$ may still be expensive when considering the cumulative time across all B&B nodes. Refinability Definition 2 addresses the choice of P such that $b^{\text{GBT},S,P} \geq b^{\text{GBT},S',P'}$.

DEFINITION 2. Given two partitions P' and P'' of set $\mathcal{T}$, we say that P' *refines* P'' if and only if $\forall \mathcal{T}' \in P'$, $\exists \mathcal{T}'' \in P''$ such that $\mathcal{T}' \subseteq \mathcal{T}''$. This definition of refinement implies a partial ordering between different partitions of $\mathcal{T}$. We express the refinement relation by $\preceq$, i.e., $P' \preceq P''$ if and only if P' refines P''.

Lemma 3 allows bound tightening by partition refinements. Its proof is similar to Lemma 2.

LEMMA 3. *Let P and P' be two partitions of $\mathcal{T}$. If $P' \preceq P$, then $b^{\text{GBT},P'} \leq b^{\text{GBT},P}$.*

In general, for two partitions P and P', we do not know a priori which partition results in a superior GBT lower bound. However, by Lemma 3, P' refining P suffices for $b^{\text{gbt},P} \geq b^{\text{gbt},P'}$. Therefore, given partition P' for the parent node, constructing P for the child node S by unifying subsets of P' will not result in inferior lower bounds.

Algorithm 2 improves $b^{\text{GBT},S',P'}$ at node S by computing a refined partition P. Suppose that $P' = \{\mathcal{T}_1, \ldots, \mathcal{T}_k\}$. Each GBT ensemble subset $\mathcal{T}' \in P'$ corresponds to a smaller subproblem with $n^{\mathcal{T}',S}$ leaves ($z_{t,l}$ variables) over the domain S. Initially, Algorithm 2

Algorithm 2 Non-Root Node Partition Refinement

P': parent node partition
Sort $P' = \{\mathcal{T}_1, \ldots, \mathcal{T}_k\}$ so that $n^{\mathcal{T}_1} \leq \ldots \leq n^{\mathcal{T}_k}$
$P \leftarrow \emptyset$
$i = 1$
while $i < \lfloor n/2 \rfloor$ and the time limit is not exceeded **do**
    $P \leftarrow P \cup \{\mathcal{T}_{2i-1} \cup \mathcal{T}_{2i}\}$
    $i \leftarrow i + 1$
end while
$P \leftarrow P \cup \{\mathcal{T}_j \in P' : j > i\}$
return P

sorts the subsets of P' in non-decreasing order of $n^{\mathcal{T}',S}$. Then, it iteratively takes the union of consecutive pairs and calculates the associated lower bound, i.e., the first calculation is for $b^{\text{GBT},S,\{\mathcal{T}_1 \cup \mathcal{T}_2\}}$, the second is for $b^{\text{GBT},S,\{\mathcal{T}_3 \cup \mathcal{T}_4\}}$ and so forth. The iterations terminate when all unions have been recalculated, or at user defined time limit q resulting in two sets of bounds: those that are combined and recalculated, and those that remain unchanged. Assuming that the final subset that is updated has index $2l$, the new partition of the trees at node S is $P = \{\mathcal{T}_1 \cup \mathcal{T}_2, \ldots, \mathcal{T}_{2l-1} \cup \mathcal{T}_{2l}, \mathcal{T}_{2l+1}, \ldots, \mathcal{T}_k\}$ with GBT bound $b^{\text{GBT},S,P} = \sum_{i=1}^{l} b^{\text{GBT},S,\{\mathcal{T}_{2i-1} \cup \mathcal{T}_{2i}\}} + \sum_{i=2l+1}^{k} b^{\text{GBT},S',\{\mathcal{T}_i\}}$. The second sum is a result of placing time limit q on updating the GBT lower bound. Time limit q maintains a balance between searching and bounding. Unifying any number of subsets satisfies Lemma 3, but Algorithm 2 unifies pairs to keep the resulting subproblems manageable.

5.2.3. Node Pruning In the B&B algorithm, each node can access: (i) the current best found feasible objective f^*, (ii) a lower bound on the convex penalties $b^{\text{cvx},S}$, and (iii) a lower bound on the GBT part $b^{\text{GBT},S}$. The algorithm prunes node S if:

$$b^{\text{cvx},S} + b^{\text{GBT},S} > f^*, \tag{5}$$

i.e., if all feasible solutions in S have objective inferior to f^*.

5.3. Branching

5.3.1. Branch Ordering Next branch selection is a critical element of B&B Algorithm 1. Each branch is a GBT split (x_i, v) choice and eliminates a certain number of GBT leaves.

Branching w.r.t. a GBT split that minimizes the number of remaining leaves may lead to a smaller number of subsequent B&B iterations by reducing the GBT size.

Computing inactive$((x_i, v), \mathcal{T})$, the number of discarded GBT leaves when branching w.r.t. $(x_i, v_{i,j})$ $\forall i \in [n], j \in [m_i]$, is time-consuming. So, we heuristically approximate the number of discarded leaves by assuming that GBTs are perfectly balanced and assign a pseudo-cost to each branch. Let $r((x_i, v), t)$ return the set of nodes in tree t that split on (x_i, v). Denote by $d(s)$ the depth of node s where the depth of the root node is 0. We initialize branch pseudo-costs, i.e., the (x_i, v) pairs, using a weighting function:

$$\text{weight}((x_i, v), t) = \sum_{s \in r((x_i, v), t)} 2^{-d(s)}, \tag{6a}$$

$$\text{weight}((x_i, v), \mathcal{T}) = \sum_{t \in \mathcal{T}} \text{weight}((x_i, v), t). \tag{6b}$$

Lemma 4 shows that the weight function and the number of inactive (excluded) leaves are proportional for a GBT instance containing only balanced trees of identical height.

LEMMA 4. *Let $\mathcal{T}$ define a GBT instance where each tree $t \in \mathcal{T}$ is balanced with depth d. Let* inactive$((x_i, v), \mathcal{T})$ *return the number of inactive leaves when branching on branch pair (x_i, v) in $\mathcal{T}$. Then* $\text{weight}((x_i, v), \mathcal{T}) = \text{inactive}((x_i, v), \mathcal{T})/2^d$.

GBT lower bounding benefits from inactivating a large number of leaves, so B&B Algorithm 1, based on Lemma 4, sorts the branch pairs in non-increasing weight order. This branching order may aid GBT bounding $b^{\text{GBT},S}$ as the algorithm descends into the search tree. The weight function characterizes split pairs that are of low depth in their individual trees and occur more often among all trees. Within a single tree, Lemma 5 shows that if two split pairs cover the same leaves, then they will be assigned the same weight.

LEMMA 5. *Given tree $t \in \mathcal{T}$, let* cover(t, s) *return the set of leaves that node $s \in t$ covers, i.e., those below s. If split pairs (x, v) and (x', v') cover the same set of leaves in tree t, i.e.,*

$$\bigcup_{s \in r((x,v),t)} \text{cover}(s, t) = \bigcup_{s \in r((x',v'),t)} \text{cover}(s, t),$$

then $\text{weight}((x, v), t) = \text{weight}((x', v'), t)$.

Figure 3 shows how split pairs may be interchangeable and, by Lemma 5, locally the weighting function will not be unfair to these split pairs.

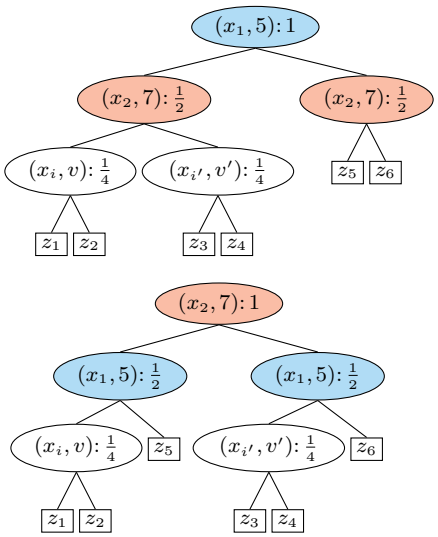

Figure 3 **Equation (6) GBT node weighting function applied to two equivalent trees. The function assigns each occurrence of split pair (x_i, v) weight $2^{d(s)}$ where s is the split node. Pairs $(x_1, 5)$ and $(x_2, 7)$ cover the same leaves. So, by Lemma 5, their summed weights are equal ensuring fairness.**

Algorithm 3 Strong Branching

S: B&B node with bounds $b^{\text{GBT},S}$ and $b^{\text{cvx},S}$
$B^S = [(x_{i_1}, v_1), \ldots, (x_{i_l}, v_l)]$: l next branches list w.r.t. Section 5.3.1 pseudo-cost order
for $(x_i, v) \in B^S$ **do**
    $S_{\text{left}}, S_{\text{right}}$: S children by branching on (x_i, v)
    Compute $b^{\text{cvx},S_{\text{left}}}$ and $b^{\text{cvx},S_{\text{right}}}$
    if $\max\{b^{\text{cvx},S_{\text{left}}}, b^{\text{cvx},S_{\text{right}}}\} + b^{\text{GBT},S} < f^*$ **then**
        return $\arg\min\{b^{\text{cvx},S_{\text{left}}}, b^{\text{cvx},S_{\text{right}}}\}, (x_i, v)$
    end if
end for
return $S, (x_{i_1}, v_1)$

5.3.2. Strong Branching Branch selection is fundamental to any B&B algorithm. *Strong branching* selects a branch that enables pruning with low effort computations and achieves a non-negligible speed-up in the algorithm's performance (Morrison et al. 2016). Strong branching increases the size of efficiently solvable large-scale mixed-integer problems and is a major solver component (Klabjan et al. 2001, Anstreicher et al. 2002, Anstreicher 2003, Easton et al. 2003, Belotti et al. 2009, Misener and Floudas 2013, Kılınç et al. 2014). Here, strong branching leverages the easy-to-solve convex penalty term for pruning.

At a B&B node S, branching produces two children S_{left} and S_{right}. Strong branching Algorithm 3 considers the branches in their Section 5.3.1 pseudo-cost ordering and assesses each branch by computing the associated convex bound. Under the strong branching test, one node among S_{left} and S_{right} inherits the convex bound $b^{\text{cvx},S}$ from the parent, while the other requires a new computation. Suppose that $S' \in \{S_{\text{left}}, S_{\text{right}}\}$ does not inherit $b^{\text{cvx},S}$. If $b^{\text{cvx},S'}$ satisfies the Equation (5) pruning condition without GBT bound improvement, then S' is immediately selected as the strong branch and strong branching repeats at the other child node S''. Figure 4 illustrates strong branching. When Algorithm 1 does not find a strong branch, it performs a GBT lower bound update and branches on the first item of the branch ordering. Algorithm 1 then adds this node's children to a set of unexplored nodes and continues with the next B&B iteration.

Strong branching allows efficient pruning when the convex objective part is significant. Strong branching may reduce the computational overhead incurred by GBT bound recalculation when Algorithm 3 selects multiple strong branches between GBT bound updates. While a single strong branch assessment is negligible, the cumulative cost of calculating convex bounds for all branches may be high. Section 5.3.1 orders the branches according to a measure of effectiveness aiding GBT bounding, so the time spent deriving strong branches with small weighting function may be better utilized in improving the GBT bound. Opposed to full strong branching, i.e., assessing all branches, strong branching Algorithm 3 uses a lookahead approach (Achterberg et al. 2005). Parameterized by a lookahead value $l \in \mathbb{Z}_{>0}$, Algorithm 3 investigates the first l branches. If Algorithm 3 finds a strong branch, Algorithm 1 repeats Algorithm 3, otherwise the B&B Algorithm 1 updates the GBT bound $b^{\text{GBT},S,P}$ at the current node. Algorithm 3 keeps strong branching checks relatively cheap and maintains a balance between searching and bounding.

5.4. Heuristics

To prune, i.e., satisfy Equation (5), consider two heuristic methods generating good feasible solutions to Problem (1): (i) a mixed-integer convex programming (convex MINLP) approach, and (ii) particle swarm optimization (PSO) (Eberhart and Kennedy 1995, Kennedy and Eberhart 1995). The former approach uses the decomposability of GBT ensembles, while the latter exploits trade-offs between the convex and objective GBT parts.

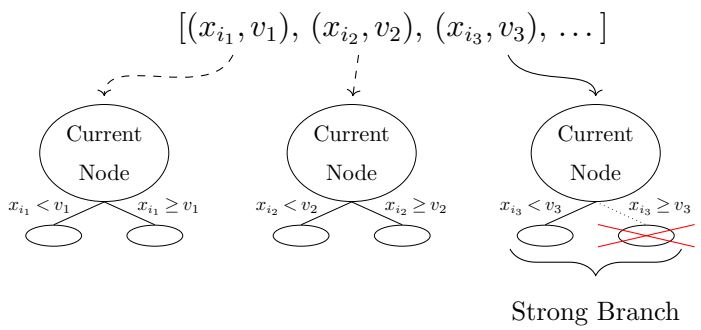

Figure 4 **Strong branching for selecting the next spatial branch. A strong branch leads to a node that is immediately pruned, based on a convex bound computation.**

5.4.1. Mixed-Integer Convex Programming Heuristic Although convex MINLP solvers provide weak feasible solutions for large-scale instances of Problem (1), they may efficiently solve moderate instances to global optimality (Westerlund and Pettersson 1995, Tawarmalani and Sahinidis 2005, Vigerske 2012, Misener and Floudas 2014, Lundell et al. 2017). For a given a subset $\mathcal{T}' \subseteq \mathcal{T}$ of trees, let $f_{\mathcal{T}'}(\cdot)$ be the objective function obtained by ignoring the trees $\mathcal{T} \setminus \mathcal{T}'$. Then, $\min_{\boldsymbol{v}^L \leq \boldsymbol{x} \leq \boldsymbol{v}^U} \{f_{\mathcal{T}'}(\boldsymbol{x})\}$ may be significantly more tractable than the original problem instance when $|\mathcal{T}'| << |\mathcal{T}|$. So, the Algorithm 4 heuristic solves the original convex MINLP by sequentially solving smaller convex MINLP sub-instances of increasing size. A sub-instance is restricted to a subset $\mathcal{T}' \subseteq \mathcal{T}$ of GBTs. Let $\mathcal{T}^{(k)}$ be the subset of trees when the k-th heuristic iteration begins. Initially, $\mathcal{T}^{(0)} = \emptyset$, i.e., $f_{\mathcal{T}^{(0)}}(\cdot)$ consists only of the convex part. Denote by $\boldsymbol{x}^{(k)}$ the sub-instance optimal solution minimizing $f_{\mathcal{T}^{(k)}}(\cdot)$. Note that $\boldsymbol{x}^{(k)}$ is feasible for the full instance. Each iteration k chooses a set of N additional trees $\mathcal{T}^{\text{next}} \subseteq \mathcal{T} \setminus \mathcal{T}^{(k)}$ and constructs $\mathcal{T}^{(k+1)} = \mathcal{T}^{(k)} \cup \mathcal{T}^{\text{next}}$, i.e., $\mathcal{T}^{(k)} \subseteq \mathcal{T}^{(k+1)}$. Consider two approaches for picking the N trees between consecutive iterations: (i) training-aware selection and (ii) best improvement selection. Termination occurs when the time limit is exceeded and Algorithm 4 returns the best computed solution.

Training-aware selection Let $T_1, T_2, \ldots, T_m$ be the tree generation order during training. This approach selects the trees $\mathcal{T}^{\text{next}}$ according to this predefined order. That is, in the k-th iteration, $\mathcal{T}^{(k)} = \{T_1, \ldots, T_{kN}\}$ and $\mathcal{T}^{\text{next}} = \{T_{kN+1}, \ldots, T_{(k+1)N}\}$. A GBT training algorithm constructs the trees iteratively, so each new tree reduces the current GBT ensemble error with respect to the training data. Thus, we expect that the earliest-generated trees better approximate the learned function than the latest-generated trees. Specifically, for two subsets $\mathcal{T}_A, \mathcal{T}_B \subseteq \mathcal{T}$ with the property that $t_a < t_b$ for each $T_{t_a} \in \mathcal{T}_A$ and $T_{t_b} \in \mathcal{T}_B$, we expect

that $|f_{\mathcal{T}_A}(\boldsymbol{x}) - f^*(\boldsymbol{x})| \leq |f_{\mathcal{T}_B}(\boldsymbol{x}) - f^*(\boldsymbol{x})|$, for each $\boldsymbol{v}^L \leq \boldsymbol{x} \leq \boldsymbol{v}^U$, where f^* is the original objective function, i.e., the optimal approximation. Intuitively, earlier trees place the GBT function within the correct vicinity, while later trees have a fine tuning role.

Best improvement selection In this approach, the k-th iteration picks the N trees with the maximum contribution when evaluating at $\boldsymbol{x}^{(k)}$. We select $\mathcal{T}^{\text{next}} \subseteq \mathcal{T} \setminus \mathcal{T}^{(k)}$ so that, for each pair of trees $T_t \in \mathcal{T}^{\text{next}}$ and $T_{t'} \in \mathcal{T} \setminus (\mathcal{T}^{(k)} \cup \mathcal{T}^{\text{next}})$, it holds that $f_t(\boldsymbol{x}^{(k)}) \geq f_{t'}(\boldsymbol{x}^{(k)})$. Assuming that approximation $\mathcal{T}^{(k)}$ is poor, then $\mathcal{T}^{\text{next}}$ contains the trees that refute optimality of $\boldsymbol{x}^{(k)}$ the most, from the perspective of $f_t(\boldsymbol{x}^{(k)})$ $t \in \mathcal{T} \setminus \mathcal{T}^{(k)}$.

Algorithm 4 Mixed-integer convex programming heuristic

1: $k \leftarrow 0$
2: $\mathcal{T}^{(k)} \leftarrow \emptyset$
3: **while** the time limit is not exceeded **do**
4: $\quad \boldsymbol{x}^{(k)} \leftarrow \underset{\boldsymbol{v}^L \leq \boldsymbol{x} \leq \boldsymbol{v}^U}{\arg\min} f_{T^{(k)}}(\boldsymbol{x})$
5: $\quad$ Choose $\mathcal{T}^{\text{next}}$ from $\left\{\mathcal{T}' \mid \mathcal{T}' \subseteq \mathcal{T} \setminus \mathcal{T}^{(k)}, |\mathcal{T}'| = \min\{N, |\mathcal{T} \setminus \mathcal{T}^{(k)}|\}\right\}$
6: $\quad \mathcal{T}^{(k+1)} \leftarrow \mathcal{T}^{(k)} \cup \mathcal{T}^{\text{next}}$
7: $\quad k \leftarrow k + 1$
8: **end while**
9: **return** $\underset{k \in \{0,\ldots,k-1\}}{\arg\min} f\left(\boldsymbol{x}^{(k)}\right)$

5.4.2. Particle Swarm Optimization Kennedy and Eberhart (1995) introduce PSO for optimizing continuous nonlinear functions. PSO computes a good heuristic solution by triggering m particles that collaboratively search the feasibility space. PSO picks the initial particle position $\boldsymbol{x}_i^{(0)}$ and search direction $\boldsymbol{v}_i^{(0)}$ of particle i randomly. The search occurs in a sequence of rounds. In round k, every particle chooses its next position $\boldsymbol{x}_i^{(k+1)}$ by following the direction specified by a weighted sum of: (i) the current trajectory direction $\boldsymbol{v}_i^{(k)}$, (ii) the particle's best found solution $\boldsymbol{p}_i$, (iii) the globally best found solution $\boldsymbol{g}$, and moving by a fixed step size. The *intertia term* $\omega \boldsymbol{v}_{\boldsymbol{i}}^{(k)}$ controls how quickly a particle changes direction. The *cognitive term* $c_1 \cdot r_1 \cdot (\boldsymbol{p}_i - \boldsymbol{x}_i^{(k)})$ controls the particle tendency to move to the best observed solution by that particle. The *social term* $c_2 \cdot r_2 \cdot (\boldsymbol{g} - \boldsymbol{x}_i^{(k)})$ controls the particle tendency to move toward the best solution observed by any particle. Coefficients ω, c_1, and

c_2 are tunable parameters. Termination occurs either when all particles are close, or within a specified time limit. Algorithm 5 in Appendix C lists the PSO algorithm.

For Problem (1), we improve the PSO performance by avoiding initial particle positions in feasible regions strictly dominated by the convex term. We project the initial random points close to regions where the GBT term is significant compared to the convex term.

6. Case Studies

Our case studies consider GBT instances where training data is not evenly distributed over the $[\boldsymbol{v}^L, \boldsymbol{v}^U]$ domain. So, while $\boldsymbol{x} \in [\boldsymbol{v}^L, \boldsymbol{v}^U]$ is feasible, GBT($\boldsymbol{x}$) may be less meaningful for $\boldsymbol{x}$ far from training data. The Problem (1) cvx($\boldsymbol{x}$) function, for the case studies, is a penalty function constructed with principal component analysis (PCA) (Jolliffe 2002).

PCA characterizes a large, high-dimensional input data set $D = \{\boldsymbol{d}^{(1)}, \ldots, \boldsymbol{d}^{(p)}\}$ with a low-dimensional subspace capturing most of the variability (James et al. 2013). PCA defines a set of n ordered, orthogonal *loading vectors*, ϕ_i, such that ϕ_i captures more variability than $\phi_{i'}$, for $i < i'$. PCA on D defines parameters $\boldsymbol{\mu}, \boldsymbol{\sigma} \in \mathbb{R}^n$ and $\boldsymbol{\Phi} = [\phi_1 \ \ldots \ \phi_n] \in \mathbb{R}^{n \times n}$, i.e., the sample mean, sample standard deviation and loading vectors, respectively. Vectors $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ standardize D since PCA is sensitive to scaling. Often, only a few ($k < n$) leading loading vectors capture most of the variance in D and $\boldsymbol{\Phi}' = [\phi_1 \ \ldots \ \phi_k]$ may effectively replace $\boldsymbol{\Phi}$. $\boldsymbol{P} = \boldsymbol{\Phi}'\boldsymbol{\Phi}'^{\top}$ defines a projection matrix to the subspace spanned by $\{\phi_1, \ldots, \phi_k\}$.

Penalizing solutions further from training data with PCA defined projection matrix $\boldsymbol{P}$:

$$\text{cvx}(\boldsymbol{x}) = \left\| (\boldsymbol{I} - \boldsymbol{P}) \, \text{diag}(\boldsymbol{\sigma})^{-1} (\boldsymbol{x} - \boldsymbol{\mu}) \right\|_2^2 \tag{7}$$

where $\boldsymbol{I}$ is the identity matrix and diag($\cdot$) is a matrix with the argument on the diagonal. Note in Equation (7) that our specific nonlinear convex penalty is a convex quadratic.

7. Numerical Results

This section compares the Section 5 algorithms to black-box solvers. Section 7.1 provides information about the system specifications and the solvers. Sections 7.2 and 7.3 investigate two GBT instances for engineering applications, namely: (i) concrete mixture design and (ii) chemical catalysis. The concrete mixture design instance is from the UCI machine learning repository (Dheeru and Karra Taniskidou 2017). The industrial chemical catalysis instance is provided from BASF. Table 2 presents information about these instances. For both instances, we model closeness to training data using the Section 6 Principal Component Analysis (PCA) approach.

Table 2 Instance Sizes

	Concrete Mixture Design	Chemical Catalysis
GBT attributes:		
Number of trees	7750	8800
Maximum depth	16	16
Number of leaves	131,750	93200
Number of x_i continuous variables	8	42
Convex MINLP (2) attributes:		
Number of $y_{i,j}$ binary variables	8,441	2,061
Number of constraints	281,073	183,791

7.1. System and Solver Specifications

Experiments are run on an Ubuntu 16.04 HP EliteDesk 800 G1 TWR with 16GB RAM and an Intel Core i7-4770@3.40GHz CPU. Implementations are in Python 3.5.3 using Pyomo 5.2 (Hart et al. 2011, 2017) for mixed-integer programming modeling and interfacing with solvers. We use CPLEX 12.7 and Gurobi 7.5.2 as: (i) black-box solvers for the entire convex MINLP (2), (ii) branch-and-bound algorithm components for solving MILP (3) instances in the Section 5.2 GBT lower bounding procedure, and (iii) heuristic components for solving convex MINLP (2) instances in the Section 5.4 convex MINLP heuristic. Note that current versions of CPLEX and Gurobi cannot solve general convex MINLP, so we would use a more general solver if we had non-quadratic penalty functions. The R package GenSA (Xiang et al. 2013) runs the Simulated Annealing (SA) metaheuristic. We provide a SA technical description (Kirkpatrick et al. 1983) in Appendix D of the Electronic Companion. The Python module PySwarms (Miranda 2018) implements the Section 5.4 Particle Swarm Optimization (PSO) metaheuristic.

We use the default CPLEX 12.7 and Gurobi 7.5.2 tolerances, i.e., relative MIP gap, integrality and barrier convergence tolerances of 10^{-4}, 10^{-5} and 10^{-8}, respectively. We use the default SA parameters. We parameterize PSO with inertia term $\omega = 0.5$, cognitive term $c_1 = 0.7$, social term $c_2 = 0.3$, 500 particles and an iteration limit of 100. Each particle takes a randomly generated point, $\boldsymbol{x}^{(0)} \in [\boldsymbol{v}^L, \boldsymbol{v}^U]$, and its projection, $\boldsymbol{x}^{(p)}$ on $\boldsymbol{P}$ and initializes at $\boldsymbol{x} = h \cdot \boldsymbol{x}^{(0)} + (1-h) \cdot \boldsymbol{x}^{(p)}$. For our tests, we use $h = 0.15$.

7.2. Concrete Mixture Design

In concrete mixture design, different ingredient proportions result in different properties of the concrete, e.g., compressive strength. The relationship between ingredients and properties is complex, so black-box machine learning is well suited for the function estimation task (Chou et al. 2011, Erdal 2013, DeRousseau et al. 2018).

Table 3 Concrete mixture design instance: Black-box solver solutions (upper bounds) by solving the entire mixed-integer convex programming (convex MINLP) model using: (i) CPLEX 12.7, (ii) Gurobi 7.5.2, (iii) Simulated Annealing (SA), and (iv) Particle Swarm Optimization (PSO), with 1 hour timeout.

CPLEX 12.7	Gurobi 7.5.2	PSO	SA
-14.2	-17.7	-88.7	**-91.3**

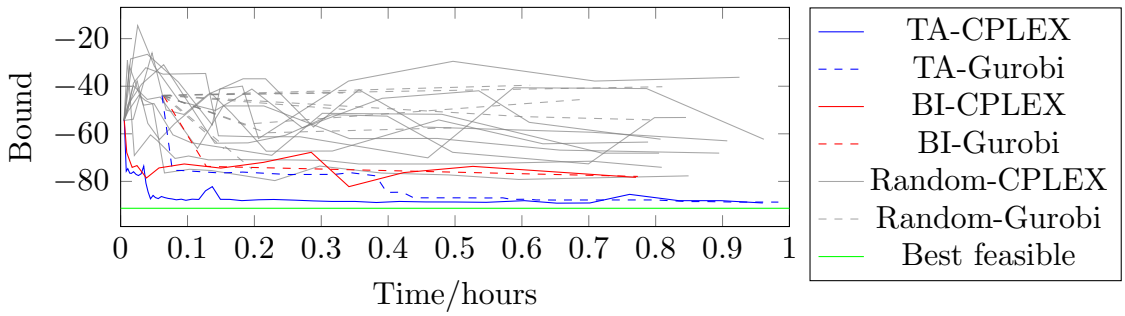

Figure 5 Concrete mixture design instance: Convex MINLP heuristic using CPLEX 12.7, or Gurobi 7.5.2 for each subproblem, and training-aware (TA), best improvement (BI), or random strategies for choosing the next trees. Each iteration selects 10 new trees. Best feasible is the Simulated Annealing solution.

7.2.1. Instance We maximize concrete compressive strength where GBTs are used for modeling. Since we maximize concrete compressive strength, negating all leaf weights $F_{t,l}$ forms an equivalent GBT instance that fits the Problem (1) minimization formulation. We use the Yeh (1998) concrete compressive strength dataset from the UCI machine learning repository (Dheeru and Karra Taniskidou 2017). This dataset has $n = 8$ continuous variables. R packages gbm (Ridgeway 2017) and caret (Kuhn 2008) are used for GBT training. Root-mean-square error is used for model selection. The resulting GBT instance has 7750 trees with max depth 16. The PCA based convex penalty has $\text{rank}(\boldsymbol{P}) = 4$, i.e., we select the first four loading vectors.

7.2.2. Heuristics Table 3 compares the CPLEX 12.7, Gurobi 7.5.2, SA, and PSO computed solutions for the entire convex MINLP, under 1 hour time limit. SA performs the best. PSO solution is relatively close to the SA best found solution, compared to CPLEX 12.7 or Gurobi 7.5.2. Figure 5 evaluates the Section 5.4.1 augmenting convex MINLP heuristic using CPLEX 12.7, Gurobi 7.5.2, and the different tree selection approaches, i.e., (i) training-aware (TA), (ii) best improvement (BI), and (iii) random selection. Figure 5 also plots the SA best-found solution. In general, both TA and BI perform better than random selection. Moreover, TA performs better than BI. Therefore, there is a benefit in choosing the earlier trees to find good heuristic solutions. Interestingly, the solution found

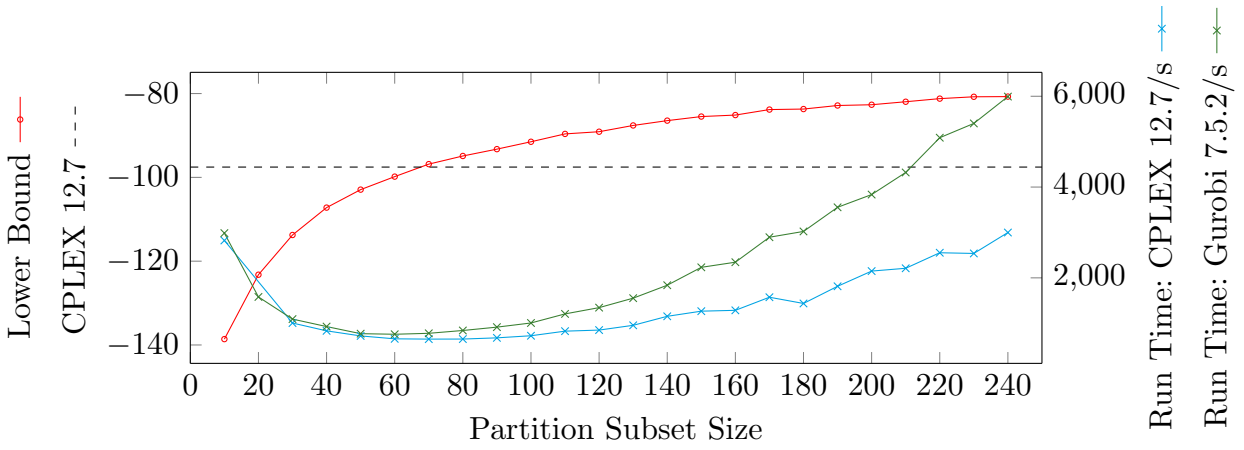

Figure 6 **Concrete mixture design instance: Global GBT lower bounding. Solid lines evaluate the Section 5.2 GBT lower bounding approach, i.e., computed lower bounds and running times, for different partition subset sizes. The dashed line plots the CPLEX 12.7 lower bound for MILP formulation (3), with 1 hour timeout. Gurobi 7.5.2 produces a lower bound -574 (not plotted).**

in the first iteration of the augmenting convex MINLP heuristic, i.e., by solely minimizing the convex part, is lower than -43, while the upper bounds reported by CPLEX 12.7 and Gurobi 7.5.2 after one hour of execution are greater than -18.

7.2.3. GBT Lower Bounding Figure 6 evaluates the Section 5.2 GBT lower bounding approach. We lower bound the GBT part of the concrete mixture design instance using different partition subset sizes. For a given partition subset size, we use either CPLEX 12.7, or Gurobi 7.5.2 to solve the MILP corresponding to each GBT subset. Figure 6 illustrates the global GBT lower bound improvement as the partition subset size increases. For the entire MILP instance, CPLEX 12.7 and Gurobi 7.5.2 achieve GBT lower bounds -97 and -547 (not plotted), respectively, within 1 hour. The Section 5.2 approach achieves a lower bound of -83 (partition size 190), in 1 hour, and improves upon the MILP solver lower bounds in less than 15 minutes (partition size 70). For small subset sizes, the partition-based lower bounding has decreasing running time because of the overhead from many sequentially solved subproblems. For larger subset sizes, the running time increases exponentially, while the lower bound improvement rate decreases exponentially.

7.2.4. Branch-and-Bound Algorithm We instantiate the branch-and-bound algorithm with the best found Table 3 feasible solution. We use either CPLEX 12.7, or Gurobi 7.5.2 in the lower bounding procedure. The root node partition subset size is 70. The non-root node lower bounding time limit is 120 seconds. Furthermore, we set the strong branching

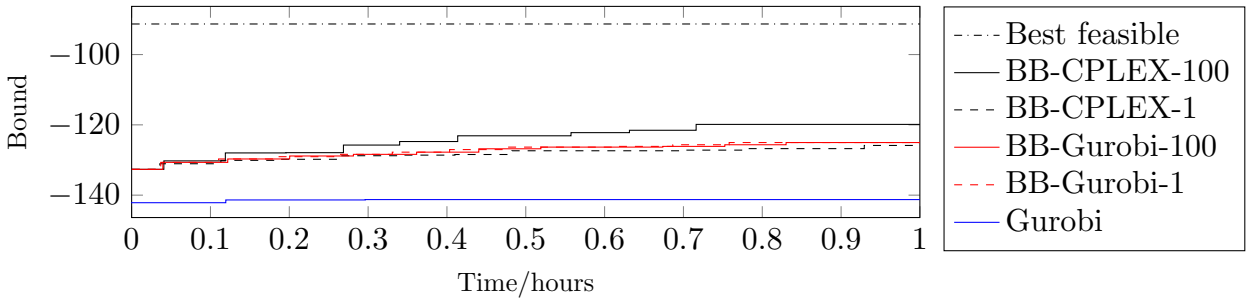

Figure 7 **Concrete mixture design instance: B&B lower bound improvement compared to Gurobi 7.5.2, with 1 hour timeout. The B&B Algorithm 1 uses either CPLEX 12.7, or Gurobi 7.5.2 in the lower bounding procedure and strong branching lookahead list size either 1, or 100. The dashed-dotted line reports best found feasible solution (upper bound).**

lookahead list size either 1, or 100. We compare the B&B results to 3 hour runs of CPLEX 12.7 and Gurobi 7.5.2 on the entire convex MINLP model, which accounts for the B&B algorithm using 1 hour for a heuristic solution, 1 hour for GBT lower bounding at the root node, and 1 hour for the B&B search.

Figure 7 depicts the lower bound improvement. The B&B algorithm lower bound improves over time, but there is still a non-negligible gap from the best-known feasible solution after 1 hour. This gap appears to be due to a cluster-like effect in the GBTs (Du and Kearfott 1994, Wechsung et al. 2014, Kannan and Barton 2017), where the breakpoints variables are quite close. In the B&B algorithm, if the current lookahead list contains these clusters, strong branching is less effective. CPLEX 12.7 results in an out-of-memory error prior to beginning the branch-and-bound search therefore its lower bounds are relatively poor. Gurobi 7.5.2 returns an incumbent of -85 and a lower bound of -141, after 2 hours, and these do not improve further in the subsequent hour. The B&B algorithm, at 2 hours, i.e., prior to tree search, has an incumbent of -91 and a lower bound not less than -133. Given an additional hour for tree search, the gap reduces further.

7.3. Chemical Catalysis

BASF uses catalysts to improve yield and operating efficiency. But, modeling catalyst effectiveness is highly nonlinear and varies across different applications. BASF has found GBTs effective for modeling catalyst behavior. Capturing the high-dimensional nature of catalysis over the entire feasible domain requires many experiments, too many to run in practice. Running a fewer number of experiments necessitates penalizing solutions further from where the GBT function is trained.

Table 4 Chemical catalysis BASF instance (with different λ values): Black-box solver solutions (upper bounds) by solving the entire mixed-integer convex programming (convex MINLP) model using: (i) CPLEX 12.7, (ii) Gurobi 7.5.2, (iii) Simulated Annealing (SA), and (iv) Particle Swarm Optimization (PSO), with 1 hour timeout.

λ	CPLEX 12.7	Gurobi 7.5.2	PSO	SA
0	*	-158.5	-96.8	**-168.2**
1	*	-101.6	-89.8	**-130.7**
10	952	-100.1	-97.6	**-102.7**
100	1,040	11.5	-82.7	**-84.2**
1000	18,579	606.5	-76.5	**-81.3**

7.3.1. Instance The BASF industrial instance contains $n = 42$ continuous variables. The convex part of the instance takes the following form:

$$\mathrm{cvx}_\lambda(\boldsymbol{x}) = \lambda \left\| (\boldsymbol{I} - \boldsymbol{P}) \operatorname{diag}(\boldsymbol{\sigma})^{-1} (\boldsymbol{x} - \boldsymbol{\mu}) \right\|_2^2 + \left(100 - \sum_{i \in \mathcal{I}^{\%}} x_i \right)^2 \tag{8}$$

The Equation (8) augend contains PCA defined parameters $\boldsymbol{P}$, $\boldsymbol{\sigma}$ and $\boldsymbol{\mu}$, and is similar to the Equation (7) penalty function. Equation (8) differs from Equation (7) in its penalty parameter $\lambda > 0$ and its right summand. A larger λ generates more conservative solutions with respect to PCA subspace $\boldsymbol{P}$. The Equation (8) addend aims to generate solutions where $x_i \in \mathcal{I}$, i.e., proportions of the chemicals being mixed, sum to 100%. The test instance has $\operatorname{rank}(\boldsymbol{P}) = 2$ and $|\mathcal{I}^{\%}| = 37$. The GBT part contains 8800 trees where 4100 trees have max depth 16, the remaining trees have max depth 4, the total number of leaves is 93,200 and the corresponding Problem (3) MILP model has 2061 binary variables.

7.3.2. Heuristics Table 4 compares the CPLEX 12.7, Gurobi 7.5.2, SA, and PSO computed solutions for the entire covex MINLP, under a 1 hour time limit. SA outperforms all others. PSO performs well for larger λ values, because it keeps the contribution of the convex part low at initialization. Gurobi 7.5.2 also performs relatively well for smaller λ values, however due to solver tolerances it may report incorrect objective values. For example, using $\lambda = 0$ the solver reports an objective of -174.1, however a manual evaluation results in -158.5. In fact, both CPLEX 12.7 or Gurobi 7.5.2, may produce incorrect outputs due to solver tolerances, hence a specialized fixing method may be necessary.

Figures 8 and 9 evaluate the Section 5.4.1 augmenting convex MINLP heuristic for different values of the λ input parameter. We investigate the augmenting convex MINLP heuristic performance using either CPLEX 12.7, or Gurobi 7.5.2 for solving convex MINLP sub-instances and each of the: (i) training-aware (TA), (ii) best improvement (BI), and (iii)

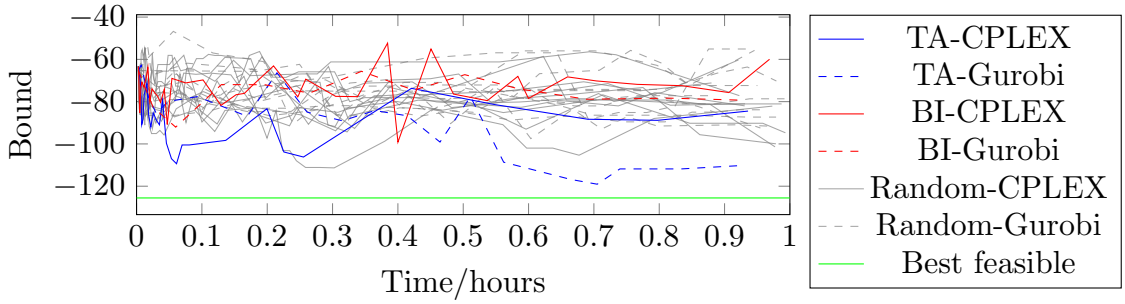

Figure 8 **Chemical catalysis instance ($\lambda = 1$): Convex MINLP heuristic using CPLEX 12.7, or Gurobi 7.5.2 for each subproblem, and training-aware (TA), best improvement (BI), or random strategies for choosing the next trees. Each iteration selects 10 new trees. Best feasible is the Simulated Annealing solution.**

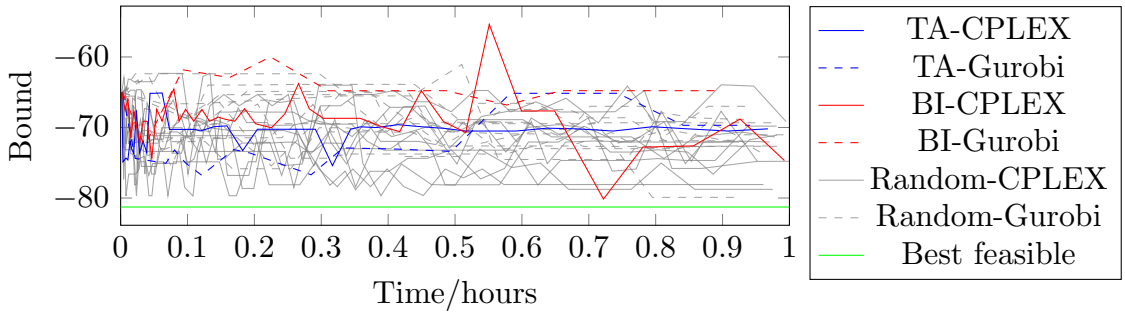

Figure 9 **Chemical catalysis instance ($\lambda = 1000$): Convex MINLP heuristic using CPLEX 12.7, or Gurobi 7.5.2 for each subproblem, and training-aware (TA), best improvement (BI), or random strategies for choosing the next trees. Each iteration selects 10 new trees. Best feasible is the Simulated Annealing solution.**

random selection strategies. The Figures 8 and 9 best feasible solution is the one produced by SA. For $\lambda = 1$, TA constructs several heuristic solutions that outperform both the BI and random selection ones. In this case, since the GBT part dominates the convex part, TA iteratively computes a better GBT approximation. For $\lambda = 1000$, TA and BI exhibit comparable performance, with BI finding the best solution. Random selection also performs well because the convex part dominates the GBT part.

7.3.3. GBT Lower Bounding Figure 10 evaluates the Section 5.2.2 GBT lower bounding approach, for different partition subset sizes. For each partition subset size, CPLEX 12.7 or Gurobi 7.5.2 computes the optimal GBT value for each subset of GBTs. The Figure 10 results resemble Figure 6. In particular, (i) the lower bound is improved with larger subset sizes, (ii) there is a time-consuming modelling overhead for solving many small MILPs for small subset sizes, and (iii) the running time increases exponentially, though non-monotonically, for larger subset sizes. We compare the lower bounding approach with solving the entire MILP (3) using CPLEX 12.7, or Gurobi 7.5.2 as black-box solvers. Our lower

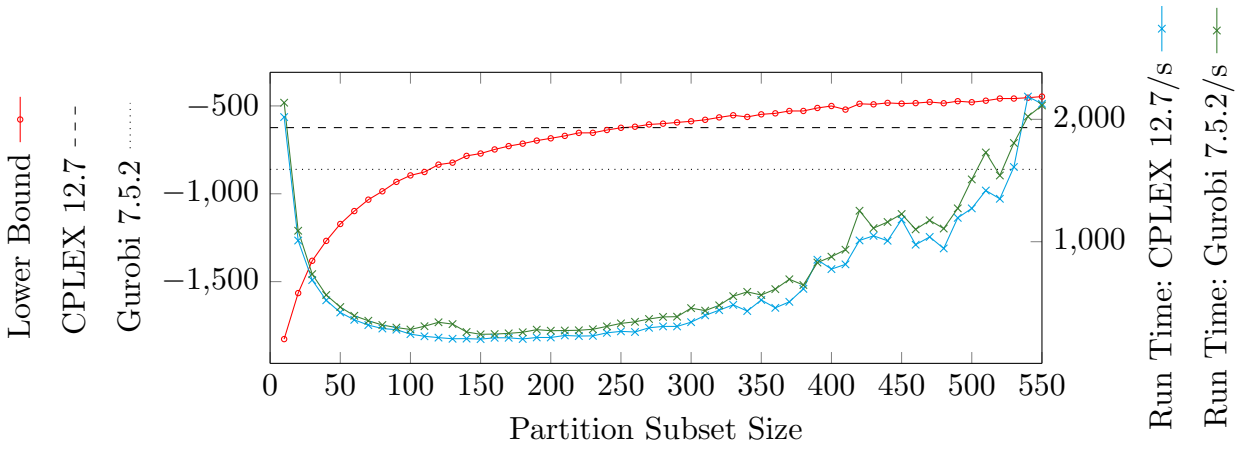

Figure 10 **Chemical catalysis BASF instance: Global GBT lower bounding. Solid lines evaluate the Section Section 5.2 GBT lower bounding approach, i.e., computed bounds and running times, for different partition subset sizes. The dashed and dotted lines plot the CPLEX 12.7 and Gurobi 7.5.2 lower bounds, respectively, for the Problem (3) MILP formulation, with 1 hour timeout.**

bounding approach exhibits a superior time-to-lower bound performance: (i) it improves the Gurobi 7.5.2 lower bound with subset size 140 and 4 minutes of execution, and (ii) it improves the CPLEX 12.7 lower bound with subset size 360 and 8 minutes of execution.

7.3.4. Branch-and-Bound Algorithm We instantiate the branch-and-bound algorithm with the Table 4 best found feasible solution. We use either CPLEX 12.7, or Gurobi 7.5.2. for solving MILP instances in the lower bounding procedure. The root node partition is set equal to 150 trees. The non-root node lower bounding time limit is fixed to 120 seconds. We use strong branching lookahead lists of size 1 and 100. We compare the B&B results to 3 hour runs of CPLEX 12.7 and Gurobi 7.5.2 for the entire convex MINLP, which accounts for the B&B algorithm using 1 hour for a heuristic solution, 1 hour for GBT lower bounding at the root node, and 1 hour for the B&B search

Figures 11 and 12 plot the lower bound improvement for $\lambda = 1$ and $\lambda = 1000$, respectively. CPLEX 12.7 reports a poor lower bound and does not find a feasible solution within 3 hours. The B&B algorithm closes a larger gap than Gurobi 7.5.2 for both $\lambda = 1$ and $\lambda = 1000$. For $\lambda = 1$, the global lower bound obtained before the tree search is tighter than the Gurobi 7.5.2 lower bound obtained in 2 hours. The B&B algorithm performs better for $\lambda = 1000$ because the convex part dominates the GBT part and strong branching is more effective. Strong branching affects the B&B algorithm performance. For $\lambda = 1000$, lookahead list size 100 closes more gap compared to lookahead list size 1. For $\lambda = 1$, larger lookahead list

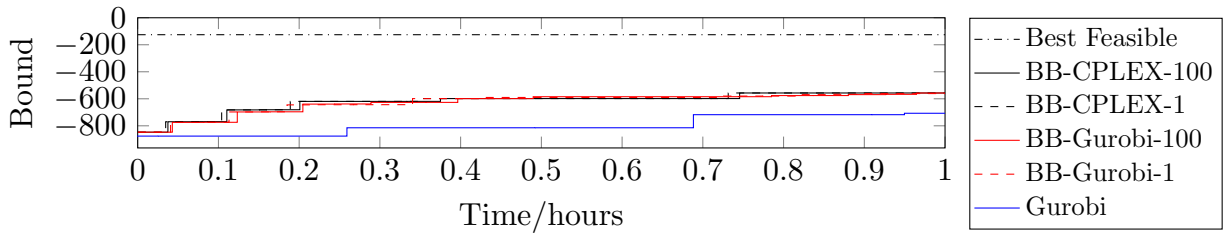

Figure 11 **Chemical catalysis BASF instance ($\lambda = 1$): B&B lower bound improvement compared to Gurobi 7.5.2 with one hour timeout. The B&B Algorithm 1 uses either CPLEX 12.7, or Gurobi 7.5.2 in the lower bounding procedure and strong branching lookahead list size either 1, or 100. The dashed-dotted line reports best found feasible solution (upper bound).**

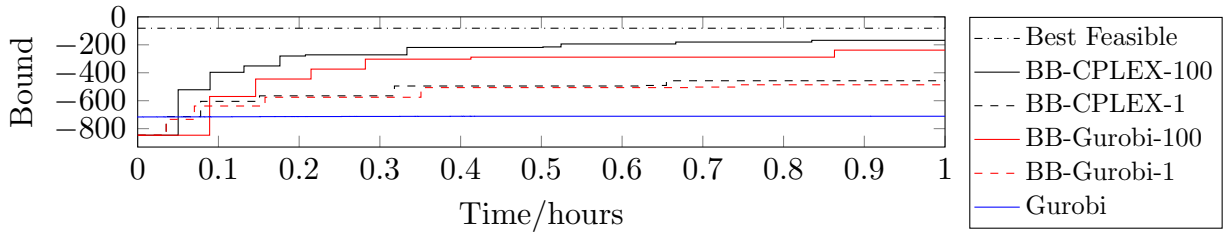

Figure 12 **Chemical catalysis BASF instance ($\lambda = 1000$): B&B lower bound improvement compared to Gurobi 7.5.2 with one hour timeout. The B&B Algorithm 1 uses either CPLEX 12.7, or Gurobi 7.5.2 in the lower bounding procedure and strong branching lookahead list size either 1, or 100. The dashed-dotted line reports best found feasible solution (upper bound).**

size does not have a noticable effect. However, this last finding does not depreciate strong branching. Since the GBT part daminates the convex aspect for small λ values, tighter GBT lower bounds might be effential for taking full advantage of strong branching.

8. Conclusion

As machine learning methods mature, decision makers want to move from solely making predictions on model inputs to integrating pre-trained machine learning models into larger decision-making problems. This paper addresses a large-scale, industrially-relevant gradient-boosted tree model by directly exploiting: (i) advanced mixed-integer programming technology with strong optimization formulations, (ii) GBT tree structure with priority towards searching on commonly-occurring variable splits, and (iii) convex penalty terms with enabling fewer mixed-integer optimization updates. The general form of the optimization problem appears whenever we wish to optimize a pre-trained gradient-boosted tree with convex terms in the objective, e.g., penalties. It would have been alternatively possible to train and then optimize a smooth and continuous machine learning model, but applications

with legacy code may start with a GBT. Our numerical results test against concrete mixture design and chemical catalysis, two applications where the global solution to an optimization problem is often particularly useful. Our methods not only generate good feasible solutions to the optimization problem, but they also converge towards proving the exact solution.

Acknowledgments

The support of: BASF SE, the EPSRC Centre for Doctoral Training in High Performance Embedded and Distributed Systems to M.M. (EP/L016796/1), and an EPSRC Research Fellowship to R.M. (EP/P016871/1).

References

Achterberg T, Koch T, Martin A (2005) Branching rules revisited. *Oper. Res. Lett.* 33(1):42–54.

Anstreicher K, Brixius N, Goux JP, Linderoth J (2002) Solving large quadratic assignment problems on computational grids. *Math. Program.* 91(3):563–588.

Anstreicher KM (2003) Recent advances in the solution of quadratic assignment problems. *Math. Program.* 97(1):27–42.

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

Belotti P, Lee J, Liberti L, Margot F, Wächter A (2009) Branching and bounds tightening techniques for non-convex MINLP. *Optim. Method. Softw.* 24(4-5):597–634.

Bertsimas D, Dunn J (2017) Optimal classification trees. *Mach. Learn.* 106(7):1039–1082.

Bertsimas D, King A (2016) OR Forum – An algorithmic approach to linear regression. *Oper. Res.* 64(1):2–16.

Bertsimas D, King A, Mazumder R (2016) Best subset selection via a modern optimization lens. *Ann. Stat.* 44(2):813–852.

Bertsimas D, Mazumder R (2014) Least quantile regression via modern optimization. *Ann. Stat.* 42(6):2494–2525.

Bixby RE (2012) A brief history of linear and mixed-integer programming computation. *Doc. Math.* 107–121.

Bollas GM, Barton PI, Mitsos A (2009) Bilevel optimization formulation for parameter estimation in vapor-liquid(-liquid) phase equilibrium problems. *Chem. Eng. Sci.* 64(8):1768–1783.

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

Boyd S, Vandenberghe L (2004) *Convex optimization* (Cambridge university press).

Breiman L, Friedman JH, Olshen RA, Stone CJ (1984) *Classification and Regression Trees* (Wadsworth, Inc.).

Chen T, Guestrin C (2016) XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794.

Chou JS, Chiu CK, Farfoura M, Al-Taharwa I (2011) Optimizing the prediction accuracy of concrete compressive strength based on a comparison of data-mining techniques. *J. Comput. Civil Eng.* 25(3):242–253.

DeRousseau M, Kasprzyk J, Srubar W (2018) Computational design optimization of concrete mixtures: A review. *Cement Concrete Res.* 109:42–53.

Dheeru D, Karra Taniskidou E (2017) UCI machine learning repository. URL `http://archive.ics.uci.edu/ml`.

Du K, Kearfott RB (1994) The cluster problem in multivariate global optimization. *J. Global Optim.* 5(3):253–265.

Easton K, Nemhauser G, Trick M (2003) Solving the travelling tournament problem: A combined integer programming and constraint programming approach. *Practice and Theory of Automated Timetabling IV*, 100–109, ISBN 978-3-540-45157-0.

Eberhart R, Kennedy J (1995) A new optimizer using particle swarm theory. *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, 39–43.

Erdal HI (2013) Two-level and hybrid ensembles of decision trees for high performance concrete compressive strength prediction. *Eng. Appl. Artif. Intel.* 26(7):1689–1697.

Fischetti M, Jo J (2018) Deep neural networks and mixed integer linear optimization. *Constraints* 23(3):296–309.

Freund Y (1995) Boosting a weak learning algorithm by majority. *Inform. Comput.* 121(2):256–285.

Friedman JH (2001) Greedy function approximation: A gradient boosting machine. *Ann. Stat.* 29(5):1189–1232.

Friedman JH (2002) Stochastic gradient boosting. *Comput. Stat. Data Anal.* 38(4):367–378.

Günlük O, Kalagnanam J, Menickelly M, Scheinberg K (2016) Optimal generalized decision trees via integer programming. *arXiv preprint arXiv:1612.03225* .

Hart WE, Laird CD, Watson JP, Woodruff DL, Hackebeil GA, Nicholson BL, Siirola JD (2017) *Pyomo–optimization modeling in Python*, volume 67 (Springer Science & Business Media), second edition.

Hart WE, Watson JP, Woodruff DL (2011) Pyomo: modeling and solving mathematical programs in Python. *Math. Program. Comput.* 3(3):219–260.

Hastie T, Tibshirani R, Friedman J (2009) *The Elements of Statistical Learning* (Springer-Verlag New York), second edition.

James G, Witten D, Hastie T, Tibshirani R (2013) *An Introduction to Statistical Learning* (Springer-Verlag New York).

Jolliffe IT (2002) *Principal Component Analysis* (Springer-Verlag New York), second edition.

Kannan R, Barton PI (2017) The cluster problem in constrained global optimization. *J. Global Optim.* 69(3):629–676.

Ke G, Meng Q, Finley T, Wang T, Chen W, Ma W, Ye Q, Liu TY (2017) LightGBM: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems 30*, 3149–3157.

Kennedy J, Eberhart R (1995) Particle swarm optimization. *Proceedings of the IEEE International Conference on Neural Networks*, volume 4, 1942–1948.

Kılınç M, Linderoth J, Luedtke J, Miller A (2014) Strong-branching inequalities for convex mixed integer nonlinear programs. *Comput. Optim. Appl.* 59(3):639–665.

Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. *Science* 220(4598):671–680, URL `http://dx.doi.org/10.1126/science.220.4598.671`.

Klabjan D, Johnson EL, Nemhauser GL, Gelman E, Ramaswamy S (2001) Solving large airline crew scheduling problems: Random pairing generation and strong branching. *Comput. Optim. Appl.* 20(1):73–91.

Kuhn M (2008) Building predictive models in R using the caret package. *J. Stat. Softw.* 28(5):1–26.

Lundell A, Kronqvist J, Westerlund T (2017) SHOT – a global solver for convex MINLP in Wolfram Mathematica. Espua A, Graells M, Puigjaner L, eds., *27th European Symposium on Computer Aided Process Engineering*, volume 40 of *Computer Aided Chemical Engineering*, 2137 – 2142 (Elsevier).

Miranda LJV (2018) PySwarms: A research toolkit for Particle Swarm Optimization in Python. *J. Open Source Softw.* 3.

Misener R, Floudas CA (2010) Piecewise-linear approximations of multidimensional functions. *J. Optim. Theory Appl.* 145(1):120–147.

Misener R, Floudas CA (2013) GloMIQO: Global Mixed-Integer Quadratic Optimizer. *J. Global Optim.* 57(1):3–50.

Misener R, Floudas CA (2014) ANTIGONE: Algorithms for continuous / integer global optimization of nonlinear equations. *J. Global Optim.* 59(2):503–526.

Misener R, Gounaris CE, Floudas CA (2009) Global optimization of gas lifting operations: A comparative study of piecewise linear formulations. *Ind. Eng. Chem. Res.* 48(13):6098–6104.

Mistry M, Callia D'Iddio A, Huth M, Misener R (2018) Satisfiability modulo theories for process systems engineering. *Comput. Chem. Eng.* 113:98–114.

Mišić VV (2017) Optimization of Tree Ensembles. *ArXiv e-prints* ArXiv:1705.10883.

Miyashiro R, Takano Y (2015) Mixed integer second-order cone programming formulations for variable selection in linear regression. *Eur. J. Oper. Res.* 247(3):721–731.

Morrison DR, Jacobson SH, Sauppe JJ, Sewell EC (2016) Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optim.* 19:79–102.

Nocedal J, Wright SJ (2006) *Sequential Quadratic Programming*, 529–562. ISBN 978-0-387-40065-5.

Ridgeway G (2017) Package 'gbm'. URL `https://cran.r-project.org/web/packages/gbm/index.html`.

Rossi F, Van Beek P, Walsh T (2006) *Handbook of constraint programming* (Elsevier).

Schweidtmann AM, Mitsos A (2018) Global deterministic optimization with artificial neural networks embedded. *arXiv preprint arXiv:1801.07114* .

Singer AB, Taylor JW, Barton PI, Green WH (2006) Global dynamic optimization for parameter estimation in chemical kinetics. *J. Phys. Chem. A* 110(3):971–976.

Snoek J, Larochelle H, Adams RP (2012) Practical Bayesian optimization of machine learning algorithms. *Advances in Neural Information Processing Systems 25*, 2951–2959.

Sra S, Nowozin S, Wright SJ (2012) *Optimization for Machine Learning* (MIT Press).

Tawarmalani M, Sahinidis NV (2005) A polyhedral branch-and-cut approach to global optimization. *Math. Program.* 103:225–249.

Vaswani N, Bouwmans T, Javed S, Narayanamurthy P (2018) Robust subspace learning: Robust pca, robust subspace tracking, and robust subspace recovery. *IEEE Signal Proc. Mag.* 35(4):32–55.

Vielma JP, Ahmed S, Nemhauser G (2010) Mixed-integer models for nonseparable piecewise-linear optimization: Unifying framework and extensions. *Oper. Res.* 58(2):303–315.

Vigerske S (2012) *Decomposition in Multistage Stochastic Programming and a Constraint Integer Programming Approach to Mixed-Integer Nonlinear Programming*. PhD in Mathematics, Humboldt-University Berlin.

Wechsung A, Schaber SD, Barton PI (2014) The cluster problem revisited. *J. Global Optim.* 58(3):429–438.

Westerlund T, Pettersson F (1995) An extended cutting plane method for solving convex MINLP problems. *Comput. Chem. Eng.* 19:131–136.

Xiang Y, Gubian S, Suomela B, Hoeng J (2013) Generalized simulated annealing for efficient global optimization: the GenSA package for R. *R J.* 5.

Yeh IC (1998) Modeling of strength of high-performance concrete using artificial neural networks. *Cement Concrete Res.* 28(12):1797–1808.

Appendix A: Full convex MINLP formulation

$$\min_{\boldsymbol{v}^L \le \boldsymbol{x} \le \boldsymbol{v}^U} \ \mathrm{cvx}(\boldsymbol{x}) + \sum_{t\in\mathcal{T}} \sum_{l\in\mathcal{L}_t} F_{t,l} z_{t,l} \tag{9a}$$

$$\text{s.t.} \sum_{l\in\mathcal{L}_t} z_{t,l} = 1, \qquad \forall t \in \mathcal{T}, \tag{9b}$$

$$\sum_{l\in \mathrm{Left}_{t,s}} z_{t,l} \le y_{i(s),j(s)}, \qquad \forall t \in \mathcal{T}, s \in \mathcal{V}_t, \tag{9c}$$

$$\sum_{l\in \mathrm{Right}_{t,s}} z_{t,l} \le 1 - y_{i(s),j(s)}, \qquad \forall t \in \mathcal{T}, s \in \mathcal{V}_t, \tag{9d}$$

$$y_{i,j} \le y_{i,j+1}, \qquad \forall i \in [n],\ j \in [m_i - 1], \tag{9e}$$

$$x_i \ge v_{i,0} + \sum_{j=1}^{m_i} (v_{i,j} - v_{i,j-1})(1 - y_{i,j}), \qquad \forall i \in [n], \tag{9f}$$

$$x_i \le v_{i,m_i+1} + \sum_{j=1}^{m_i} (v_{i,j} - v_{i,j+1}) y_{i,j}, \qquad \forall i \in [n], \tag{9g}$$

$$y_{i,j} \in \{0,1\}, \qquad \forall i \in [n],\ j \in [m_i], \tag{9h}$$

$$z_{t,l} \ge 0, \qquad \forall t \in \mathcal{T},\ l \in \mathcal{L}_t. \tag{9i}$$

Appendix B: Omitted Proofs

LEMMA 4. *Let $\mathcal{T}$ define a GBT function where each $t \in \mathcal{T}$ is balanced with maximum depth d. Let* $\mathrm{inactive}((x_i, v), \mathcal{T})$ *return the number of inactive (unreachable) leaves when branching on split pair (x_i, v) in $\mathcal{T}$. Then,* $\mathrm{weight}((x_i, v), \mathcal{T}) = \mathrm{inactive}((x_i, v), \mathcal{T})/2^d$.

Proof Consider the set $r((x_i, v), t)$ of GBT nodes that split on (x_i, v) in tree $t \in \mathcal{T}$. For a node $s \in r((x_i, v), t)$, let $d(s)$ be the depth of s, i.e., the length (in number of edges) of the path from the root to s. Furthermore, let $w(s) = 2^{-d(s)}$ and $i(s) = 2^{d-d(s)}$ be the number of inactive leaves below s when branching w.r.t. (x_i, v). Hence $w(s) = i(s)/2^d$ and

$$\begin{aligned}
\mathrm{weight}((x_i, v), \mathcal{T}) &= \sum_{t\in\mathcal{T}} \mathrm{weight}((x_i, v), t) \\
&= \sum_{t\in\mathcal{T}} \sum_{s\in r((x_i,v),t)} w(s) \\
&= \sum_{t\in\mathcal{T}} \sum_{s\in r((x_i,v),t)} i(s)/2^d \\
&= \mathrm{inactive}((x_i, v), \mathcal{T})/2^d
\end{aligned}$$

□

LEMMA 5. *For some tree $t \in \mathcal{T}$, let cover(t, s) be the set of leaves covered by node $s \in t$. If the set of leaves covered by split pairs (x_i, v) and $(x_{i'}, v')$ in t are equal, i.e.,*

$$\bigcup_{s\in r((x_i,v),t)} \mathit{cover}(t, s) = \bigcup_{s\in r((x_{i'},v'),t)} \mathit{cover}(t, s),$$

then $\mathrm{weight}((x_i, v), t) = \mathrm{weight}((x_{i'}, v'), t)$.

Proof The result holds trivially for $(x_i, v) = (x_{i'}, v')$. Assume that $(x_i, v) \neq (x_{i'}, v')$. Since (x_i, v) and $(x_{i'}, v')$ cover the same leaves in t, every node that splits w.r.t. (x_i, v) is either ancestor or descendant of some node that splits w.r.t. $(x_{i'}, v')$, and vice versa. Let $t(s)$ denote the subtree of t rooted at node s. As the term $\text{weight}((x_i, v), t)$ sums the weights of all nodes that split w.r.t. (x_i, v), w.l.o.g., it suffices to show that $\text{weight}((x_i, v), t(s)) = \text{weight}((x_{i'}, v'), t(s))$, $\forall\ s \in r((x_i, v), t)$ such that $r((x_{i'}, v'), t(s)) \neq \emptyset$. A symmetric statement follows by swapping (x_i, v) and $(x_{i'}, v')$.

Claim 1 shows that there exists a subtree rooted at some node $\hat{s} \in t(s)$ such that both child nodes of $\hat{s}$, namely s' and s'', split on $(x_{i'}, v')$. Denote by $d(s', t(s))$ the depth of node s' at tree $t(s)$.

CLAIM 1. *Let* $d_{\max}((x_{i'}, v'), t(s)) = \max\{d(s', t(s)) | s' \in r((x_{i'}, v'), t(s))\}$ *be the maximum depth of a node* s' *in the subtree* $t(s)$ *that splits on* $(x_{i'}, v')$*. So, consider such a node* $s' \in r((x_{i'}, v'), t(s))$ *such that* $\text{depth}(s', t(s)) = d_{\max}((x_{i'}, v'), t(s))$*. For the sibling node* s'' *of* s'*, i.e.,* $s'' \neq s'$ *and* $\text{parent}(s'', t(s)) = \text{parent}(s', t(s))$*, it holds that* $s'' \in r((x_{i'}, v'), t(s))$*.*

Proof Assume for a contradiction that $s'' \notin r((x_{i'}, v'), t(s))$. Because s' is of maximum depth branching on $(x_{i'}, v')$, it must be the case that $r((x_{i'}, v'), t(s'')) = \emptyset$. Since (x_i, v) covers all leaves in $t(s)$, there exists node $\tilde{s} \in r((x_{i'}, v'), t(s))$ that is an ancestor of s'' and therefore an ancestor of s' which contradicts that $s' \in r((x_{i'}, v'), t(s))$. $\square$

Claim 2 shows that $t \in \mathcal{T}$ is equivalent, in terms of tree evaluations and weights, to the tree t' obtained from t by swapping the nodes splitting on (x_i, v) and $(x_{i'}, v')$ so that $(x_{i'}, v')$ always occurs higher.

CLAIM 2. *Consider a node* $s \in t$ *splitting on* (x_i, v) *whose child nodes both split on* $(x_{i'}, v')$*. Let* $a, b \in \{\text{left}, \text{right}\}$ *and denote by* s_a, s_{ab} *the child nodes following the* a *branch from* s *and the* b *branch from* s_a*, respectively. Define a new tree* t' *as identical to* t *up to nodes not in* $t(s)$*, set the node located at* s *in* t' *to split on* $(x_{i'}, v')$*, and call this node* s'*. Analagously define* s'_a*,* s'_{ab} *and* $t'(s'_{ab})$*. Set* s'_a *to split on* (x_i, v) *and* $t'(s'_{ab}) = t(s_{ba})$*. Then,* $\text{GBT}_t(\boldsymbol{x}) = \text{GBT}_{t'}(\boldsymbol{x})$*, for each* $\boldsymbol{x} \in [\boldsymbol{L}, \boldsymbol{U}]$*, and* $\text{weight}((x_i, v), t) = \text{weight}((x_i, v), t')$*, for every split pair* (x_i, v) *with* $v = v_{i,j}$ *such that* $i \in [j]$ *and* $j \in [m_i]$*.*

Proof We first show equivalence of the trees. Fix $\boldsymbol{x} \in [\boldsymbol{L}, \boldsymbol{U}]$. Let $V_t(\boldsymbol{x})$ and $V_{t'}(\boldsymbol{x})$ be the sets of GBT nodes on the root-to-leaf path that evaluates $\boldsymbol{x}$ in trees t and t' respectively. If $s \notin V_t(\boldsymbol{x})$ then the relevant root-to-leaf paths of $\boldsymbol{x}$ are identical in trees t and t', so $\text{GBT}_t(\boldsymbol{x}) = \text{GBT}_{t'}(\boldsymbol{x})$. Assume now that $s \in V_t(\boldsymbol{x})$. Since t and t' are identical up to s and s', respectively, it must be the case that $s' \in V_{t'}(\boldsymbol{x})$. By construction, the fact that $s_{ab} \in S$ implies that $s'_{ba} \in S'$. Furthermore $t(s_{ab}) = t'(s'_{ba})$. Hence, the two sequences of nodes after s_{ab} and s'_{ba}, respectively, are identical and $\text{GBT}_t(\boldsymbol{x}) = \text{GBT}_{t'}(\boldsymbol{x})$.

We now show that the weights are the same. Let V_t and $V_{t'}$ be the sets of split nodes in trees t and t', respectively. Clearly there exists a bijective mapping $f : V_t \setminus \{s, s_{\text{left}}, s_{\text{right}}\} \to V_{t'} \setminus \{s', s'_{\text{left}}, s'_{\text{right}}\}$ such that, for each $s \in V_t \setminus \{s, s_{\text{left}}, s_{\text{right}}\}$, it holds that $d(s, t) = d(f(s), t')$, and $s \in t$ splits on the same pair with $f(s) \in t'$. Therefore, for each $(x_{i''}, v'') \notin \{(x_i, v), (x_{i'}, v')\}$ such that $v'' = v_{i'',j''}$ with $i'' \in [n]$ and $j'' \in [m_i]$, $\text{weight}((x_{i''}, v''), t) = \text{weight}((x_{i''}, v''), t')$. For (x_i, v), let w be the contribution to $\text{weight}((x_i, v), t)$ from nodes in $V_t \setminus \{s\}$. By the bijective mapping f, the contribution to $\text{weight}((x_i, v), t')$ from nodes in $V_{t'} \setminus \{s'_{\text{left}}, s'_{\text{right}}\}$

is also w. It remains taking into account nodes s, s'_{left} and s'_{right}. Taking into account the node depths and the construction of t':

$$\text{weight}((x_i, v), t) = w + 2^{-d(s,t)} = w + 2^{-d(s,t)-1} + 2^{-d(s,t)-1} = \text{weight}((x_i, v), t').$$

With a similar argument for $(x_{i'}, v')$, Claim 2 follows. □

With repeated application of Claims 1 and 2, we can define a finite sequence $T^{(k)}$ of k trees where all trees are equivalent to $t(s)$ and the final element t' of $T^{(k)}$ has root node spitting on $(x_{i'}, v')$. This tree t' assigns weight 1 to both (x_i, v) and $(x_{i'}, v')$. Therefore (x_i, v) and $(x_{i'}, v')$ have the same weight in $t(s)$, and Lemma 5 follows. □

Appendix C: Particle Swarm Optimization

Algorithm 5 lists the particle swarm optimization algorithm (Kennedy and Eberhart 1995).

Algorithm 5 Particle Swarm Optimization

Compute initial position $\boldsymbol{x}_{\boldsymbol{i}}^{(0)} \in \mathbb{R}^n$ and velocity $\boldsymbol{v}_{\boldsymbol{i}}^{(0)} \in \mathbb{R}^n$ for each particle $i = 1, \ldots, m$.
$\boldsymbol{p_i} \leftarrow \boldsymbol{x}_{\boldsymbol{i}}^{(0)}$
$\boldsymbol{g} \leftarrow \arg\min\{f(\boldsymbol{p_i})\}$
$k \leftarrow 0$
while the time limit is not exceeded **do**
  for $i = 1, \ldots, m$ **do**
    Choose random values $r_1, r_2 \sim U(0, 1)$
    $\boldsymbol{v}_{\boldsymbol{i}}^{(k+1)} \leftarrow \omega \boldsymbol{v}_{\boldsymbol{i}}^{(k)} + c_1 \cdot r_1 \cdot (\boldsymbol{p_i} - \boldsymbol{x}_{\boldsymbol{i}}^{(k)}) + c_2 \cdot r_2 \cdot (\boldsymbol{g} - \boldsymbol{x}_{\boldsymbol{i}}^{(k)})$
    $\boldsymbol{x}_{\boldsymbol{i}}^{(k+1)} \leftarrow \boldsymbol{x}_{\boldsymbol{i}}^{(k)} + \boldsymbol{v}_{\boldsymbol{i}}^{(k+1)}$
    if $f(\boldsymbol{x}_{\boldsymbol{i}}^{(k+1)}) < f(\boldsymbol{p_i})$ **then**
      $\boldsymbol{p_i} \leftarrow \boldsymbol{x}_{\boldsymbol{i}}^{(k+1)}$
    end if
  end for
  $\boldsymbol{g} \leftarrow \arg\min\{f(\boldsymbol{p_i})\}$
  $k \leftarrow k + 1$
end while

Appendix D: Simulated Annealing

Algorithm 6 lists the simulated annealing algorithm (Kirkpatrick et al. 1983).

Algorithm 6 Simulated Annealing

Compute an initial solution $\boldsymbol{x}^{(0)} \in \mathbb{R}^n$.
Set initial temperature $T^{(0)} = 1$ and probability constant $c = 1$.
Set temperature factor $\alpha \in [0.80, 0.99]$.
$t = 0$, $k = 0$
while $T^{(t)} > \epsilon$ **do**
    for r iterations **do**
        Select a neighboring solution $\boldsymbol{x} \in \mathcal{N}(\boldsymbol{x}^{(k)})$ randomly.
        if $f(\boldsymbol{x}) < f(\boldsymbol{x}^{(k)})$ **then**
            $\boldsymbol{x}^{(k+1)} \leftarrow \boldsymbol{x}$
            $k \leftarrow k + 1$
        else
            Choose $p \sim U(0, 1)$
            if $\exp(-(f(\boldsymbol{x}) - f(\boldsymbol{x}^{(k)}))/cT^{(t)}) > p$ **then**
                $\boldsymbol{x}^{(k+1)} \leftarrow \boldsymbol{x}$
                $k \leftarrow k + 1$
            end if
        end if
    end for
    $T^{(t+1)} \leftarrow \alpha T^{(t)}$
    $t \leftarrow t + 1$
end while

Appendix E: Table of Notation

Name	Description
GBT Ensemble Definition	
n	Number of the GBT-trained function (continuous) variables
i	Continuous variable index
x_i	Continuous variable
$\boldsymbol{x}$	Vector $(x_1, \ldots, x_n)^T$
$\mathcal{T}$	Set of gradient boosted trees
t	Gradient boosted tree
$\mathcal{V}_t$	Set of split nodes (vertices) in tree t
$\mathcal{L}_t$	Set of leaf nodes in tree t
s	Split node associated with a tree t and mainly referred to as (t, s)
$i(t, s)$	Continuous variable index associated with split node s in tree t
$v(t, s)$	Splitting value of variable $x_{i(t,s)}$ at split node s in tree t

$\text{GBT}_t(\boldsymbol{x})$	Tree t evaluation at point $\boldsymbol{x}$
$\text{GBT}(\boldsymbol{x})$	GBT ensemble evaluation at point $\boldsymbol{x}$
Convex MINLP with GBTs Problem Definition	
$\text{cvx}(\boldsymbol{x})$	Convex function evaluation at point $\boldsymbol{x}$
m_i	Number of variable x_i splitting values
$v_{i,j}$	j-th greatest variable x_i splitting value
v_i^L or $v_{i,0}$	Variable x_i lower bound
v_i^U or v_{i,m_i+1}	Variable x_i upper bound
$\boldsymbol{v}^L$	Vector $(v_1^L, \ldots, v_n^L)$
$\boldsymbol{v}^U$	Vector $(v_1^U, \ldots, v_n^U)$
$\text{Left}_{t,s}$	Set of leaves in the subtree rooted in the left child of s in tree t
$\text{Right}_{t,s}$	Set of leaves in the subtree rooted in the right child of s in tree t
$F_{t,l}$	Contribution of leaf node l in tree t
$y_{i,j}$	Binary variable indicating whether $x_i \leq v_{i,j}$, or not
$z_{t,l}$	Binary variable specifying whether tree t evaluates at leaf l
d	Maximum tree depth
Branch-and-Bound Algorithm Overview	
$[\boldsymbol{v}^L, \boldsymbol{v}^U]$	Optimization problem global domain
$S = [\boldsymbol{L}, \boldsymbol{U}]$	Optimization problem subdomain / B&B node
(x_i, v)	GBT splitting point / B&B branch
$S_{\text{left}}, S_{\text{right}}, S_c, S'$	B&B nodes
Q	Set of unexplored B&B nodes
P_{root}	Initial GBT ensemble partition at B&B root node
P, P', P''	GBT ensemble partitions
$b^{\text{cvx},S}$	Convex lower bound over domain S
$b^{\text{GBT},S,P}$	GBT lower bound over domain S w.r.t. partition P
Lower Bounding	
R^S	Optimal objective value, i.e., tightest relaxation
$\hat{R}^S$	Relaxation dropping linking constraints
$b^{\text{GBT},S,*}$	Optimal GBT lower bound over domain S
$\boldsymbol{x}^*$	Optimal solution
i, j, l	Subset indices of a GBT ensemble partition
k	GBT ensemble partition size
$\mathcal{T}_i, \mathcal{T}_j, \mathcal{T}', \mathcal{T}''$	Subsets of GBTs
N	GBT ensemble subset size
$n^{\mathcal{T},S}$	Number of leaves in GBT subset $\mathcal{T}$ over domain S
f^*	Best found feasible objective
q	Time limit on lower bound improvement algorithm
Branching	
B	Branch ordering
$r((x_i, v), t)$	Set of nodes in tree t that split on (x_i, v)
$d(s)$	Depth of split node s (root node has zero depth)
$w(s)$	Weight of split node s
$i(s)$	Number of inactive leaves below split s when branching w.r.t. (x_i, s)
$\text{weight}((x_i, v), t)$	Weight assigned to (x_i, v) in tree t
$\text{weight}((x_i, v), \mathcal{T})$	Weight assigned to (x_i, v) in GBT ensemble $\mathcal{T}$
$\text{inactive}((x_i, v), \mathcal{T})$	Number of inactive leaves when branching on pair (x_i, v) in $\mathcal{T}$
$\text{cover}(t, s)$	Set of leaves covered by split node s at tree t
$S, S_{\text{left}}, S_{\text{right}}, S_0$	B&B nodes denoted by their corresponding domain
l	Strong branching lookahead parameter

Table 5: Nomenclature