Optimization with Gradient-Boosted Trees and Risk Control

Miten Mistry 1  Dimitrios Letsios 1  Ruth Misener 1  Gerhard Krennrich 2  Robert M. Lee 2

Abstract

Decision trees effectively represent the sparse, high dimensional and noisy nature of chemical data from experiments. Having learned a function from this data, we may want to thereafter optimize the function, e.g., picking the best chemical process catalyst. In this way, we may repurpose legacy predictive models. This work studies a large-scale, industrially-relevant mixed-integer quadratic optimization problem involving: (i) gradient-boosted pre-trained regression trees modeling catalyst behavior, (ii) penalty functions mitigating risk, and (iii) penalties enforcing composition constraints. We develop heuristic methods and an exact, branch-and-bound algorithm leveraging structural properties of gradient-boosted trees and penalty functions. We numerically test our methods on an industrial instance.

1. Introduction

Machine learning traditionally focusses on model predictivity and the computational expense of training. 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). Here, we investigate optimization problems where the pretrained model prediction appears in the objective function. Modeling of physical and chemical processes is a major industrial example, where not only we require tools to predict the outcome of certain operating conditions, but also guidance towards a better set of operating conditions with respect to some performance criteria.

One option is using smooth and continuous machine learning models, e.g., neural networks and support vector machines with radial basis function kernels, to represent these physical and chemical processes. The resulting optimization models may be addressed using local nonlinear optimization methods (Nocedal & Wright, 2006). But feasible solutions come without a quantifiable guarantee of globality, even when using multi-start algorithms to escape local minima. The value of global optimization is known in the chemical processing literature (Boukouvala et al., 2016), e.g., local minima can lead to infeasible parameter estimation (Singer et al., 2006) or misinterpreted chemical 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.

Also, although many machine learning techniques deal with feature correlation in training data, careful formulation of the subsequent optimization problem is essential to avoid implicit and unintentional feature causal roles. Consider, for instance, using historical data from a manufacturing process for quality maximization. The data may exhibit correlation between two process parameters: the temperature and the concentration of a chemical additive. A machine learning model of the system assigns weights to the parameters for future predictions. Lacking additional information, numerical optimization may produce candidate solutions with temperature and concentration combinations that are drastically different from past observations. For instance, the machine learning model may incorrectly associate temperature as responsible for an observed effect. A remedy to control the optimizer’s adventurousness is to include a term penalizing deviation from the training data subspace. Large values of this risk control parameter generate conservative solutions. Smaller values of the penalty term explores regions with greater possible rewards at the cost of additional uncertainty.

This work elaborates on optimization methods for problems whose objective includes gradient-boosted tree (GBT) model predictions (Friedman, 2001; Hastie et al., 2009) and risk terms to penalize deviation from the covariance structure of training data. Advantages of GBTs are myriad and justify their prominence among the winners in machine learning competitions (Chen & Guestrin, 2016; Ke et al., 2017). GBTs are robust to scale differences in the training data features, handle easily both categorical and numeri-
Optimization with Gradient-Boosted Trees and Risk Control

cal variables, and can minimize arbitrary differentiable loss functions. In addition, much work has been done to accelerate the training of GBTs using graphical processing units and distributed computing resources (Zhang et al., 2017).

Our approach formulates a mixed-integer quadratic programming (MIQP) problem whose objective consists of a discrete GBT trained function and a continuous convex penalty function. Our goal is to design exact optimization methods computing either globally optimal solutions, or solutions within a quantified distance from the global optimum. We formulate the problem as an MIQP and solve a large industrial instance using commercial approaches. Motivated by weak numerical findings, we develop a novel branch-and-bound method that exploits the tree ensemble combinatorial structure and the penalty function convexity in an integrated setting. Numerical analysis substantiates the strength of our approach.

The paper proceeds as follows. Section 2 formally defines the optimization problem. Section 3 formulates the problem as an MIQP and performs worst cases analysis. Section 4 develops heuristics. Section 5 presents our branch-and-bound method. Section 6 presents a numerical analysis on a large-scale industrial instance. Finally, Section 7 concludes.

2. Optimization Problem

We analyze an optimization problem that consists of penalty functions and gradient-boosted tree (GBT) trained functions (Friedman, 2001; 2002). GBTs are a subclass of boosting methods (Freund, 1995). Boosting methods convert a collection of weak learners into a strong learner, where a weak learner is at least better than random guessing. For GBTs, the weak learners are classification and regression trees (Breiman et al., 1984).

In this work, our analysis is restricted to GBTs that only consist of regression trees, i.e., no categorical variables. Each level of a GBT splits variable $x_i$ into $x_i < v$ and $x_i \geq v$ where $v$ is some constant defined at training. A GBT trained function is a collection of binary trees, each of which provides its own independent contribution when evaluating at $x$. The overall output is the sum of all tree evaluations. For a given $x$, a tree evaluation follows a root-to-leaf path by querying $x_j < v$ or $x_j \geq v$ and following the left or right child, respectively, in each tree level. The leaf that corresponds to $x$ contains the tree’s contribution.

Assume that we want to optimize a GBT trained function. Since the GBT function approximates an unknown function, we may trust an optimal solution close to regions with many training points. To better approximate the remaining regions, we add a convex penalty term whose parameters are obtained by principal component analysis (PCA) on the training data. We consider the following optimization problem:

$$\min_{u^L \leq x \leq u^U} \text{cvx}_\lambda(x) + \text{GBT}(x),$$

where

$$\text{cvx}_\lambda(x) = \lambda \left( (I - P) \Sigma^{-1} (x - \mu) \right)^2 + \left( 100 - \sum_{i \in \mathcal{T}^\%} x_i \right)^2,$$

and $x = (x_1, \ldots, x_n)^T$ is the variable vector. GBT($x$) is the GBT trained function value at $x$. Penalty parameter $\lambda \in \mathbb{R}_{\geq 0}$, identity matrix $I \in \mathbb{R}^{n \times n}$, projection matrix $P \in \mathbb{R}^{n \times n}$, mean $\mu \in \mathbb{R}^n$, and diagonal matrix of standard deviations $\Sigma \in \mathbb{R}^{n \times n}$ define the convex penalty term. Variable subset $\mathcal{T}^\% \subseteq \{1, \ldots, n\}$ contains variables $x_i$ that should sum close to 100%. Parameters $u^L, u^U \in \mathbb{R}^n$ enforce box constraints that lower and upper bound variable $x$. Table 1 defines the model sets, parameters and variables.

The additional term, which encourages the subset $\mathcal{T}^\% \subseteq \{1, \ldots, n\}$ of variables $x_i$ to sum close to 100%, is an artifact of the chemical processing application. We expect that the mass fraction of chemical compositions sums to 100 and, in the application, this term in the objective is equivalent to a constraint as there are several inert chemicals, i.e., variables $x_i$ that participate in the mass fraction sum but do not appear in the GBTs. But, more generally, our method can include any convex penalty term.

The problem instances that this paper addresses consist of a sum of independently trained GBT functions. However, without loss of generality, we equivalently optimize a single GBT function which is the union of all original GBTs.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_i^L, v_i^U$</td>
<td>Lower and upper bound of variable $x_i$</td>
</tr>
<tr>
<td>$x_i$</td>
<td>Continuous variable, $i \in {1, \ldots, n}$</td>
</tr>
</tbody>
</table>

For convex part:

- $\lambda \geq 0$ PCA penalty parameter
- $I$ $n \times n$ identity matrix
- $P$ $n \times n$ PCA projection matrix
- $\mu$ Vector of means
- $\Sigma$ $n \times n$ diagonal matrix of standard deviations
- $i \in \mathcal{T}^\%$ Indices that sum to 100%

For gradient boosted part:

- $t \in \mathcal{T}$ 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 $t$
- $v_{i,j}$ Variable $i$’s $j$-th breakpoint
- $F_{i,t}$ 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 $(t, l)$
3. Mixed-Integer Quadratic Formulation

Equation (1) consists of a continuous penalty 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 quadratic programming (MIQP) formulation. The main ingredient of the MIQP 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 MIQP is:

\[
\begin{align}
\min & \quad \text{cvx}_{\lambda}(x) + \text{GBT MILP objective} \\
\text{s.t.} & \quad \text{GBT MILP constraints}, \\
& \quad \text{Variable linking constraints}.
\end{align}
\]

MILP approaches for machine learning show competitive performance for some important applications (Bertsimas & Mazumder, 2014; Bertsimas & King, 2016; Bertsimas et al., 2016; Bertsimas & Dunn, 2017; Miyashiro & Takano, 2015). The performance of MILP methods arises from the significant improvement of mixed-integer algorithms (Bixby, 2012) and the availability of commercial codebases.

3.1. GBT MILP Formulation

We form the GBT MILP using the Mišić (2017) approach. Figure 1 shows how a GBT partitions the domain \([u^L, u^U]\) of \(x\). Optimizing a GBT function reduces to optimizing the leaf selection, i.e., finding an optimal interval, opposed to a specific \(x\) value. Aggregating over all GBT split nodes produces a vector of ordered breakpoints \(v_{i,j}\) for each \(x_i\) variable: \(v_{i,j}^L = v_{i,0} < v_{i,1} < \cdots < v_{i,m_i} < v_{i,m_i+1} = v_{i,j}^U\). Consecutive pairs of breakpoints define a set of intervals where the GBT function is constant. Each point \(x_i \in [v_{i,j}^L, v_{i,j}^U]\) 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 leaf nodes in the subtrees rooted in the left and right children of split node \(s\), respectively.

The GBT problem is formulated by Eq. (3). Equation (3a) minimizes the total value of the active leaves. Equation (3b) selects exactly one leaf per tree. Equations (3c) and (3d) enforce that a leaf is activated only if all corresponding splits occur. Equation (3e) ensures that if \(x_i \leq v_{i,j-1}\), then \(x_i \leq v_{i,j}\). Without loss of generality, we may drop variable \(z_{t,l}\) integrality constraint because any feasible assignment of \(y\) specifies a single leaf, i.e., a single region in Fig. 1.

\[
\begin{align}
\min & \quad \sum_{t \in \mathcal{T}} \sum_{l \in \mathcal{L}_t} F_{t,l} z_{t,l}, \\
\text{s.t.} & \quad \sum_{l \in \mathcal{L}_t} z_{t,l} = 1, \quad \forall t \in \mathcal{T}, \\
& \quad \sum_{l \in \text{Left}_{t,s}} z_{t,l} \leq y_{i(s),j(s)}, \quad \forall t \in \mathcal{T}, s \in \mathcal{V}_t, \\
& \quad \sum_{l \in \text{Right}_{t,s}} z_{t,l} \leq 1 - y_{i(s),j(s)}, \quad \forall t \in \mathcal{T}, s \in \mathcal{V}_t, \\
& \quad y_{i,j} \leq y_{i,j+1}, \quad \forall i \in [n], j \in [m_i - 1], \\
& \quad y_{i,j} \in [0, 1], \quad \forall i \in [n], j \in [m_i], \\
& \quad x_i \geq v_{i,0} + \sum_{j=1}^{m_i} (v_{i,j} - v_{i,j-1})(1 - y_{i,j}), \\
& \quad x_i \leq v_{i,m_i+1} + \sum_{j=1}^{m_i} (v_{i,j} - v_{i,j+1})y_{i,j},
\end{align}
\]

for all \(i \in [n]\). Note that we express the linking constraints using non-strict inequalities to avoid computational issues when optimizing with strict inequalities. Combining Eqs. (2) to (4) defines the mixed-integer quadratic programming (MIQP) formulation of Eq. (1).

3.2. Linking Constraints

Equations (4a) and (4b) relate the continuous \(x_i\) variables to the binary \(y_{i,j}\) variables as follows:

\[
\begin{align}
x_i & \geq v_{i,0} + \sum_{j=1}^{m_i} (v_{i,j} - v_{i,j-1})(1 - y_{i,j}), \\
x_i & \leq v_{i,m_i+1} + \sum_{j=1}^{m_i} (v_{i,j} - v_{i,j+1})y_{i,j},
\end{align}
\]

for all \(i \in [n]\). 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. However, not
all leaf combinations of leaves are valid evaluations. In a consistent leaf combination where one leaf enforces \( x_i < v_1 \) and another enforces \( x_i \geq v_2 \), it must be the case that \( v_2 < v_1 \). Let \( d \) be maximum depth of a tree in \( T \). Then, the number of leaf combinations is upper bounded by \( 2^{(d-1)|T|} \). Since the number of feasibility checks for a single combination is \( \frac{1}{2}|T|(|T|-1) \), an upper bound on the total number of feasibility checks is \( 2^{d-1}|T|^2(|T|-1) \). Among others, this observation implies that the worst case performance of an exact method improves as the number of trees decreases.

4. Heuristics

We propose heuristic methods that generate good feasible solutions to our optimization problem based on two approaches: (i) mixed-integer quadratic programming (MIQP), and (ii) particle swarm optimization (PSO) (Eberhart & Kennedy, 1995; Kennedy & Eberhart, 1995). The former approach is motivated by the decomposability of GBT ensembles, while the latter one exploits trade-offs between the convex part and the GBT part in the objective function.

MIQP-based heuristic. While commercial MIQP solvers provide weak feasible solutions for large-scale Eq. (1) problem instances, they may efficiently solve moderate instances to global optimality. This observation motivates the computation of heuristic solutions by decomposing an MIQP instance into smaller MIQP sub-instances. A sub-instance is restricted to a subset \( T' \subseteq T \) of GBTs. Our MIQP-based heuristic solves sub-instances iteratively. Let \( T_i \) be the subset of trees when the \( i \)-th heuristic iteration begins. Initially, \( T_0 = \emptyset \). At each iteration, subset \( T_i \) is computed by the union of \( T_{i-1} \) and \( N \) additional trees in \( T \setminus T_{i-1} \), i.e., \( T_{i-1} \subseteq T_i \). Denote by \( x^{(i)} \) the sub-instance optimal solution for the subset \( T_i \) of trees, e.g., \( x^{(0)} \) is the optimal solution computed by solely optimizing the convex part.

We propose two approaches for picking the subsequent \( N \) trees. Our first approach selects the trees in \( T_i \setminus T_{i-1} \) according to the order in which the trees are generated during the training process. The trees are constructed iteratively one by one and each new tree aims in minimizing the GBT ensemble error with respect to the training data. Therefore, a subset of early generated trees is expected to provide a better approximation of the GBT trained function compared to the one computed by a subset among the latest generated trees. In our second approach, the \( i \)-th iteration selects the \( N \) trees with the maximum contribution when evaluating at \( x^{(i-1)} \). These trees are expected to tighten the sub-instance approximation the most.

PSO-based heuristic. PSO computes a good heuristic solution by triggering \( N \) particles that collaboratively search the feasibility space. We pick the initial particle positions randomly. The search occurs in a sequence of rounds. In each round, every particle chooses its next position following the direction specified by a weighted sum of (i) the globally best found solution, (ii) the particle’s best found solution, (iii) the direction of its current trajectory, and moving by a fixed step size. Termination occurs either when all particles are close, or within a specified time limit. A key observation is that we should avoid initializing the particle positions in feasible regions strictly dominated by the convex term in Eq. (1). Furthermore, we should ensure sufficiently large initial distances between the particles. We improve the PSO performance by selecting random points and projecting them relatively close to regions where the convex term does not strictly dominate the GBT term. The projected points are the initial particle positions.

5. Branch-and-Bound Algorithm

Branch-and-bound is an exact method used for solving mixed-integer nonlinear optimization problems. Using a divide-and-conquer principle, branch-and-bound forms a tree of subproblems to search the domain of feasible solutions. Key aspects of branch-and-bound are: (i) rigorous lower (upper) bounding methods for the minimization (maximization) subproblems, (ii) choosing the next branch and (iii) generating good feasible solutions. In the worst case, branch-and-bound enumerates all solutions, but generally it avoids complete enumeration by pruning subproblems, i.e., removing infeasible subproblems or nodes with lower bound larger than the best found feasible solution (Morrison et al., 2016). This section exploits spatial branching that splits on continuous variables and is primarily used for handling nonconvexities, e.g., see Belotti et al. (2013).

5.1. Overview

The branch-and-bound algorithm spatially branches over the \([v^L, v^U]\) domain. It selects a variable \( x_i \), a point \( c \) and splits interval \([v^L_i, v^U_i]\) into intervals \([v^L_i, c] \) and \([c, v^U_i]\). Each of these intervals correspond to an independent subproblem and a new branch-and-bound node. At a given node, denote the reduced domain by \( S = [L, U] \). The spatial branching choice is a critical issue and arises from the relationship of the continuous \( x \) variables and Section 3 binary \( y \) variables (any feasible \( y \) is an interval of \([v^L, v^U]\)). A subproblem over \([L, U] \subset [v^L, v^U]\) immediately fixes some of the \( y \). To avoid redundant branches, all GBT define the branch-and-bound branching points.

The remainder of this section is structured as follows. Section 5.2 discusses a lower bound for the Eq. (1) optimiza-
tion problem. Section 5.3 generates an ordering to the GBT node splits to aid subproblem lower bounding. Section 5.4 explains how the branch-and-bound algorithm leverages strong branching for relatively cheap node pruning.

5.2. Lower Bounding

Effective lower bounding is fundamental to any branch-and-bound algorithm. The MIQP consists of a convex (penalty) part and an mixed-integer linear (GBT) part, handling these parts independently forms the basis of our lower bounds.

Consider assessing over domain $S = [L, U] \subseteq [v^L, v^U]$. Equation (1) restricted to $x \in S$ results in optimal objective value $R^S$, the tightest relaxation, i.e., lower bound. But finding $R^S$ is difficult, so we lower bound $R^S$ with:

$$
\hat{R}^S = \min_{x \in S} \text{cvx}_{\lambda}(x) + \min_{x \in S} \text{GBT}(x).
$$

Equation (5) treats the two Eq. (1) objective terms as independent, i.e., $\hat{R}^S$ separates the convex part from the GBT part. A mixed-integer model for Eq. (5) consists of Eqs. (2) and (3), i.e., without the Eq. (4) linking constraints. From a computational perspective, the Eq. (5) separation leverages the easy-to-solve convex part and the availability of commercial codes for the MILP GBT part. In general, $\hat{R}^S \leq R^S$, but if $S$ only corresponds to a single leaf for each GBT, $\hat{R}^S = R^S$. Tractability is an important when lower bounding. For $\hat{R}^S$, finding $\hat{R}^S$ is easy, but finding $\hat{R}^S$ is NP-hard (Mišić, 2017). With the aim of tractability, we calculate a relaxation on $\hat{R}^S$, at the expense of tightness.

5.2.1. Bound for GBTs

When evaluating the GBT functions at a given $x$, each tree provides its own independent contribution, 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 derives lower bounds on $\hat{b}^{\text{GBT},S,*}$. The most naive bound that this approach achieves is $\hat{b}^{\text{GBT},\text{naive}} = \sum_{i \in T} \min_{T \subseteq T, T_i \cap T_j = \emptyset} \{F_{i,j} | l \in L_i\}$, i.e., ignore all splits. Lower bounds $\hat{b}^{\text{GBT},S,*}$ and $\hat{b}^{\text{GBT},\text{naive}}$ of the GBT part represent two extremes: the former is generally intractable and tight whereas the latter is tractable and typically weak. We improve on $\hat{b}^{\text{GBT},\text{naive}}$ while aiming to retain tractability by considering a partition, $P$, of the set of trees, i.e., $P = \{T_1, \ldots, T_k\}$ where $T_i \subseteq T$, $T_i \cap T_j = \emptyset$ for $i \neq j$, and $\bigcup_{i=1}^k T_i = T$. Let $\hat{b}^{\text{GBT},P}$ be the minimum GBT objective value restricted to the subset $T_i$ of trees. For each $i \in [k]$, this term can be computed using Eq. (3) formulation and we set $\hat{b}^{\text{GBT},P} = \sum_{i=1}^k \hat{b}^{\text{GBT},P}$, where the term $\hat{b}^{\text{GBT},P}$ denotes the resulting lower bound using partition $P$. We have $\hat{b}^{\text{GBT},S,*} \geq \hat{b}^{\text{GBT},P} \geq \hat{b}^{\text{GBT},\text{naive}}$ for any partition $P$ of $T$.

At a given node, since we are dealing with the reduced domain $x \in S$, we may improve on $\hat{b}^{\text{GBT},\text{naive}}$ by only considering reachable leaves. Moreover, we may improve on $\hat{b}^{\text{GBT},P}$ by 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.

The branch-and-bound algorithm chooses an initial partition $P^\text{root}$ at the root node with subsets of size $N$. Our choice of $P^\text{root}$ for the instance in Section 6 has been decided numerically. The important factors to consider are the tree depth, the number of continuous variable splits and their relation with the number of binary variables for a subset of size $N$.

5.2.2. Local Lower Bound Improvement

Calculating the updated bound $\hat{b}^{\text{cvx},S}$ in a branch-and-bound node is computationally easy. Furthermore, at least one among two sibling nodes has the same convex bound with the parent node. Therefore, a single computation suffices to determine the convex bounds of two sibling nodes.

On the GBT side, the branch-and-bound algorithm initially calculates a global lower bound with partition $P^\text{root}$. When the algorithm is at some non-root branch-and-bound node with domain $S = [L, U] \subseteq [v^L, v^U]$, some of the $y_{i,j}$ variables are fixed. Since the number of valid GBT node decreases as the algorithm descends lower in the branch-and-bound tree, a tighter GBT bound may be computed by reducing the number of subsets $k$ in the partition, without very significant overhead. However, this reduction should occur with moderation because, while fixing binary variables simplifies the local problem, a complete recalculation may still carry too much of an expense when considering the cumulative use of time across all subproblems.

Consider a node $S$ with a known valid bound $\hat{b}^{\text{GBT},P'}$ for the parent node. This bound as well as the corresponding individual lower bounds $\hat{b}^{\text{GBT},T'_i}$ are also valid for node $S$. In general, for two partitions $P$ and $P'$ we do not know a priori which partition results in a superior GBT lower bound. However, if $P$ and $P'$ are such that every $T' \in P'$ is contained in some $T \in P$, then $\hat{b}^{\text{GBT},P} \geq \hat{b}^{\text{GBT},P'}$. 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.

To improve upon $\hat{b}^{\text{GBT},P'}$ at node $S$, we first sort the subsets of $P'$ in non-decreasing order with respect to the number of non-fixed variables corresponding to $S$. Let this ordering be $P' = \{T_1, T_2, \ldots, T_k\}$. Then, we iteratively take the union of consecutive pairs and calculate the associated lower bound, i.e., the first calculation
is for $b_{gbt,T_1 \cup T_2}$ and the second is for $b_{gbt,T_1 \cup T_2}$ and so forth. The iterations terminate at a user defined time limit 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 $P = \{T_1 \cup T_2, \ldots, T_{2l-1} \cup T_{2l}, T_{2l+1}, \ldots, T_k\}$ with GBT bound $b_{gbt,P} = \sum_{i=1}^{l} b_{gbt,T_{2i-1}} + \sum_{i=2l+1}^{k} b_{gbt,T_i}$. The second sum is a result of having the time limit on updating the GBT lower bound. This time limit is necessary to maintain a balance between searching and bounding.

5.2.3. Node Pruning

In the branch-and-bound algorithm, each node has access to: (i) the current best found feasible objective $f^*$, (ii) a lower bound on the convex penalty $b_{cvx,S}'$, and (iii) a lower bound on the GBT part $b_{gbt,S}$. The algorithm prunes node $S$ if $b_{cvx,S}' + b_{gbt,S} > f^*$. This condition is valid since the left hand side tells us that all feasible solutions in $S$ have objective inferior to $f^*$.

5.3. Branch Ordering

The performance of the branch-and-bound algorithm depends on its ability to prune nodes. The pruning condition depends on the quality of the best found feasible solution $f^*$ and the tightness of $b_{gbt,S}$. Bound $b_{cvx,S}'$ is tightly computed. A feasible solution may be improved by local search at a branch-and-bound node. We tighten $b_{gbt,S}$ using the Section 5.2 lower bounding method whose performance depends on how many binary variables are fixed at node $S$. Branching decisions should be effective at fixing binary variables in order to improve $b_{gbt,S}$ computation.

Since all branches of the branch-and-bound algorithm are splits from the GBTs, the GBT lower bounding step benefits from branching choices that can fix a larger number of binary variables. We order the splits by preprocessing the GBTs. Each split node contains a split pair $(x_i, v)$ and has depth $d$. Root node has $d = 0$. As Fig. 2 shows, each split node assigns to its $(x_i, v)$ pair a weight $2^{-d}$. A given pair may repeat in a single tree and over different trees. Hence, we sum up all its weights. Sorting the split pairs in non-increasing weight order defines the branch ordering.

The choice of $2^{-d}$ favors split pairs that are higher in their respective GBTs. Branching on such a split pair should be more effective at fixing binary variables. Furthermore, since we sum the weights, the branch ordering also prioritizes pairs occurring more often. This weighting also incorporates a fairness property. Assume a GBT where the root splits on $(x_{i_1}, v_1)$, while both children split on $(x_{i_2}, v_2)$. Then, these split pairs are interchangeable, i.e., $(x_{i_2}, v_2)$ could be the root and $(x_{i_1}, v_1)$ the two children. Therefore, they should have the same weight. This idea extends locally to a non-root node. Choosing a local weight of $2^{-d}$ covers this case since the weights are summed up and the GBTs are binary trees. Figure 2 shows this interchangeability property for an unbalanced tree, where the bottom tree gives a more balanced structure than the top tree.

5.4. Strong Branching

Branch selection is fundamental to any branch-and-bound 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 is known to have increased the size of efficiently solvable large-scale mixed-integer problems and is a major component of commercial solvers (Klabjan et al., 2001; Anstreicher et al., 2002; Anstreicher, 2003; Easton et al., 2003; Belotti et al., 2009; Kilinc et al., 2014). We use strong branching to leverage the easy-to-solve convex penalty term for pruning.

At a branch-and-bound node $S$, branching leads in two children $S'$ and $S''$. One node among $S'$ and $S''$ inherits the convex bound $b_{cvx,S}'$ from the parent, while the other requires a new computation. We employ strong branching by investigating the $k$ first branches in the list generated according to Section 5.3. Without loss of generality, suppose that $S'$ does not inherit $b_{cvx,S}'$. If one of the $k$ possible branches results in $b_{cvx,S}''$ that satisfies the pruning condition without GBT bound improvement, then it is immediately selected as the strong branch and we proceed with node $S''$. Figure 3 illustrates strong branching. If a strong branch cannot be found, the algorithm performs a local GBT lower bound improvement (Section 5.2.2). If the new local GBT lower bound is not large enough to prune
Figure 3. 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.

Table 2. Best feasible (BF) solution and lower bound (LB) found by MIQP solvers CPLEX 12.7 and Gurobi 6.0.3 compared with the GenSA simulated annealing solution (SA) with one hour timeout for the industrial instance of Eq. (1).

<table>
<thead>
<tr>
<th>λ</th>
<th>CPLEX 12.7</th>
<th>Gurobi 6.0.3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BF</td>
<td>LB</td>
</tr>
<tr>
<td>0</td>
<td>157,678</td>
<td>-84</td>
</tr>
<tr>
<td>1</td>
<td>1,109</td>
<td>-63</td>
</tr>
<tr>
<td>10</td>
<td>952</td>
<td>-773</td>
</tr>
<tr>
<td>100</td>
<td>1,040</td>
<td>-835</td>
</tr>
<tr>
<td>1000</td>
<td>18,579</td>
<td>-846</td>
</tr>
</tbody>
</table>

the current node, the algorithm branches on the first item of the branch ordering, it adds the children to the list of unexplored nodes and continues with the next iteration. The above cases ensure the selection of some branch.

Strong branching allows efficient pruning in regions where the contribution of the convex part in the objective is significant. Such a branch selection allows early node pruning in the branch-and-bound tree and avoids a large number of useless node repetitions. Furthermore, it reduces the computational overhead incurred by bound recalculation. Finally, it results in a sharper bound tightening at a local level.

6. Numerical Analysis

This section compares the Sections 4 and 5 algorithms against black-box solvers. The algorithms are implemented in Python 3.5.3 using Pyomo 5.2 (Hart et al., 2011; 2017) for mixed-integer modeling and solver interfacing. We use CPLEX 12.7 and Gurobi 6.0.3. 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.

We test our algorithms on an industrial instance. The convex part has \( n = 42 \) continuous variables, \( \text{rank}(P) = 2 \) 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 Eq. (3) model has 2061 binary variables. This instance was originally tackled using GenSA simu-

Table 3. Mixed-integer quadratic programming (MIQP), particle swarm optimization (PSO) and simulated annealing (SA) results with a 1 hour timeout.

<table>
<thead>
<tr>
<th>λ</th>
<th>SA</th>
<th>MIQP</th>
<th>PSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-168.2</td>
<td>-144.0</td>
<td>-150.7</td>
</tr>
<tr>
<td>1</td>
<td>-130.7</td>
<td>-104.0</td>
<td>-111.9</td>
</tr>
<tr>
<td>10</td>
<td>-102.7</td>
<td>-89.6</td>
<td>-92.0</td>
</tr>
<tr>
<td>100</td>
<td>-84.2</td>
<td>-83.2</td>
<td>-85.3</td>
</tr>
<tr>
<td>1000</td>
<td>-80.2</td>
<td>-80.4</td>
<td>-76.9</td>
</tr>
</tbody>
</table>

6.1. Mixed-Integer Quadratic Programming

We feed the instance to MIQP solvers CPLEX 12.7, Gurobi 6.0.3 and GenSA simulated annealing package with 1 hour timeout. Table 2 lists the obtained results. SA finds better heuristic solutions than the MIQP solvers. In two cases, CPLEX does not even report a feasible solution. The large optimality gap returned by the MIQP solvers implies that (i) either there exist significantly better solutions than the ones computed by SA, or (ii) the solver lower bounds are extremely weak. Determining a better optimality gap motivates the design of our branch-and-bound algorithm.

Table 2 shows that the optimization problem is difficult for state-of-the-art commercial MIQP solvers. Moreover, the problem becomes harder as parameter \( \lambda \) decreases and GBTs’ contribution to the objective becomes more significant. This finding motivates the design of optimization methods that aim in bounding efficiently the GBT trained function and exploit bounds on the convex penalty function when its contribution becomes significant. In this last case, we are equipped with stronger bounds.

6.2. Heuristics

We compare the Section 4 heuristics to simulated annealing (SA). For MIQP, each iteration introduces \( N = 10 \) new trees. Table 3 compares the heuristics for increasing \( \lambda \). SA outperforms the other heuristics for smaller \( \lambda \)’s and is competitive for larger values. Since both MIQP and PSO exploit the convex part, they both perform better for larger values of \( \lambda \). Figure 4 shows the heuristic improvement over time for \( \lambda = 0 \) and \( \lambda = 1000 \). MIQP objective improvements are not regular and drop sharply when they improve. Particle swarm optimization (PSO) converges to an inferior solution for both \( \lambda \)’s. Even though particles are initially projected towards low penalty regions, they suffer from bias, e.g., if a non-optimal solution remains the best found for many iterations, the other particles become biased towards it. For large \( \lambda \)’s, the Section 4 heuristics do not significantly improve on the SA solutions. This sug-
Optimization with Gradient-Boosted Trees and Risk Control

Figure 4. Mixed-integer quadratic programming (MIQP), particle swarm optimization (PSO) and simulated annealing (SA) results. Top: $\lambda = 0$. Bottom: $\lambda = 1000$.

Figure 5. GBT part bounding using the partition based algorithm (Section 5.2). The left axis corresponds to the lower bound and the right axis to the cumulative runtime of CPLEX 12.7/Gurobi 6.0.3 to solve the individual MILPs. The horizontal lines are the CPLEX 12.7 and Gurobi 6.0.3 lower bounds after solving the entire instance with one hour time limit.

Figure 6. Branch-and-bound lower bound improvement for $\lambda = 0$ (top) and $\lambda = 1000$ (bottom) compared to Gurobi 6.0.3 given a one hour timeout. We test the branch-and-bound algorithm with (BB-BO) and without (BB-WBO) branch ordering. BF is the best known feasible solution.

6.3. Branch-and-Bound

The Section 5 branch-and-bound algorithm initializes with a feasible solution and a GBT part global lower bound. We initialize with the best feasible solutions from Section 6.2. For the GBT lower bound, we use the Section 5.2 approach. We first investigate how different subset sizes affect the calculated bound and runtime for this algorithm.

Figure 5 compares the GBT bounding algorithm for various partition subset sizes to MILP solvers CPLEX 12.7 and Gurobi 6.0.3. The bounding algorithm calculates tighter bounds as the subset size increases. For larger sizes, the runtime shows exponential increase. Small subset sizes have a larger computational cost due to the non-negligible modeling overhead of solving many small MILPs. Comparing with a black-box approach, we see that a well chosen subset size achieves superior time-to-bound performance. A subset size of 140 solves in 4 minutes and improves on the Gurobi 6.0.3 bound. Choosing a subset size of 360 improves on the CPLEX 12.7 bound and takes 8 minutes.

Given the Fig. 5 results, we initialize the branch-and-bound algorithm using a subset size of 150 for $P_{\text{root}}$ as it is in the trough of runtime plot. Figure 6 compares the branch-and-bound algorithm with (BB-BO) and without (BB-WBO) the branch ordering to Gurobi 6.0.3 (solving the entire MIQP problem) and the best known feasible solution for $\lambda = 0$ and $\lambda = 1000$. BB-BO outperforms all of the other approaches, providing a significant decrease in the optimality gap for $\lambda = 1000$. Comparing with Gurobi 6.0.3, the $\lambda = 1000$ case produces a much better optimality gap. For $\lambda = 0$, all solvers have a fairly large optimality gap, this is due to the problem being closer to the NP-hard Eq. (3) model. Also strong branching is less likely to be invoked in early nodes of the branch-and-bound tree resulting more regular GBT bounds updates and therefore a large increase in cumulative bounding time. Both $\lambda$’s show the importance of the branch ordering. BB-BO is able to produce better bounds earlier and for $\lambda = 1000$ strong branching coupled with the branch ordering results in significant bounds improvement.

7. Conclusion

As machine learning methods mature and their predictive modeling power becomes well-respected industrially, decision makers want to move from solely making predictions on model inputs to deciding algorithmically what is the best model input. In other words, we must move towards optimizing pre-trained machine learning models. This pa-
per effectively 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 particular model application is in chemical catalysis, but the general form of the optimization problem will appear whenever we wish to optimize a pre-trained gradient-boosted tree with convex penalty terms. 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. Additionally, it is well known in catalysis that the global solution to an optimization problem is often particularly useful. The methods in this paper not only generate good feasible solutions to the optimization problem, but they also converge towards proving the exact solution.

Acknowledgements

The support of: BASF SE, the EPSRC Centere 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) is gratefully acknowledged.

References


Optimization with Gradient-Boosted Trees and Risk Control


