Quant-BnB: A Scalable Branch-and-Bound Method for Optimal Decision Trees with Continuous Features

Rahul Mazumder [1 2] **Xiang Meng** [1] **Haoyue Wang** [1]

Abstract

Decision trees are one of the most useful and popular methods in the machine learning toolbox. In this paper, we consider the problem of learning optimal decision trees, a combinatorial optimization problem that is challenging to solve at scale. A common approach in the literature is to use greedy heuristics, which may not be optimal. Recently there has been significant interest in learning optimal decision trees using various approaches (e.g., based on integer programming, dynamic programming)—to achieve computational scalability, most of these approaches focus on classification tasks with binary features. In this paper, we present a new discrete optimization method based on branch-and-bound (BnB) to obtain optimal decision trees. Different from existing customized approaches, we consider both regression and classification tasks with continuous features. The basic idea underlying our approach is to split the search space based on the quantiles of the feature distribution—leading to upper and lower bounds for the underlying optimization problem along the BnB iterations. Our proposed algorithm `Quant-BnB` shows significant speedups compared to existing approaches for shallow optimal trees on various real datasets.

1. Introduction

A decision tree is a classic machine learning predictive tool with a flowchart-like structure that allows users to derive interpretable decisions. Combined with its effectiveness in solving classification and regression tasks, it is an immensely useful tool in domains where interpretability is of great importance. Despite its appeal, the task of learning an optimal decision tree (with smallest training error) is $\mathcal{NP}$-hard (Laurent & Rivest, 1976) and thus computationally challenging. Therefore, greedy methods such as CART (Breiman et al., 1984) and ID3 (Quinlan, 1986) are popular choices. They construct decision trees through a top-down approach. Starting from the root node, data is iteratively split into subsets according to local objectives. In spite of high efficiency, greedy heuristics may not lead to an optimal solution, possibly resulting in suboptimal predictive performance (Bertsimas & Dunn, 2017).

In recent years, there have been major advances in exploring optimization methods to construct *optimal*[1] decision trees. Verwer & Zhang (2019); Günlük et al. (2021) explore mixed integer programming (MIP) approaches to learn optimal trees with a fixed depth. Aglin et al. (2020); Demirović et al. (2020) propose interesting dynamic programming (DP) approaches for optimal decision trees. Despite impressive methodological advances, optimal tree-learning approaches face the following challenges: (**i**) MIP formulations appear to have limited scalability. Verwer & Zhang (2019) report that a MIP solver cannot solve an optimal tree of depth 2 with less than 1000 observations and 10 features in 10 minutes. (**ii**) Most state-of-the-art algorithms (Demirović et al., 2020; Aghaei et al., 2021; McTavish et al., 2021) consider datasets with binary features (i.e, every feature is $\{0, 1\}$) rather than continuous ones. Algorithms for optimal trees with continuous features are much less developed[2]—our goal in this paper is to bridge this gap in the literature.

In this work we take a step towards addressing the aforementioned shortcomings by developing a novel branch-and-bound (BnB) algorithm for the computation of shallow optimal trees (e.g. depth=2, 3). In contrast to earlier approaches, our algorithm can handle both classification and regression tasks and is designed to directly handle continuous features (including a mix of continuous and binary features). In a nutshell, our proposed algorithm `Quant-BnB` utilizes

[1]Operations Research Center, MIT, Cambridge, USA [2]Sloan School of Management and Center for Statistics, MIT, Cambridge, USA. Correspondence to: Xiang Meng <mengx@mit.edu>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

[1]In this paper, *optimal* refers to a global optimal solution to the optimization problem associated with learning a decision tree.

[2]While it is possible to convert continuous features to binary features, this may result in a large number of features, which leads to large computation times—see our experiments for details. To achieve scalability, it may be more beneficial to have tailored approaches for continuous features.

the quantiles of the feature distribution to decompose the search space into sub-regions—these are subsequently used to generate lower bounds and upper bounds on the optimal value, and prune sub-optimal regions of the search-space. To our knowledge, `Quant-BnB` is the first standalone method (i.e, does not rely on proprietary optimization solvers) for optimal classification/regression trees that directly applies to datasets with continuous features. We show that `Quant-BnB` computes the optimal solution (cf. Section 4.3), and achieves significant empirical improvements compared to existing methods (cf Section 6). A Julia implementation of our code is open-sourced on GitHub[3].

Paper	*Opt*	*Feat*	*Task*	Model
Carreira-Perpiñán, 2018	✗	✓	*R/C*	TAO
Bertsimas & Dunn, 2019	✓	✓	*R/C*	MIP
Verwer & Zhang, 2019	✓	✓	*C*	MIP
Aglin et al., 2020	✓	✗	*C*	DP&BnB
Demirović et al., 2020	✓	✗	*C*	DP
Lin et al., 2020	✓	✗	*C*	DP&BnB
Aghaei et al., 2021	✓	✗	*R/C*	MIP
Our approach	✓	✓	*R/C*	BnB

Table 1. Related works on decision tree optimization. *Opt* indicates if the approach finds an optimal tree. *Feat* indicates whether the method works for continuous features (without binarizing features). *Task* indicates what tasks the method can handle: *R* for regression and *C* for classification. TAO is an alternating optimization-based heuristic. The optimal methods are based on BnB, MIP (optimization solvers) and DP.

Related Work: Various optimization techniques have been explored to learn high-quality decision trees (Bennett & Blue, 1996; Dobkin et al., 1997; Nijssen & Fromont, 2007; Farhangfar et al., 2008; Nijssen & Fromont, 2010; Carreira-Perpinán & Tavallali, 2018). A number of recent works explore MIP-approaches for optimal decision trees. For example, Bertsimas & Dunn (2019) formulate learning optimal trees with a fixed depth as a MIP model. Günlük et al. (2021) design an improved model with much fewer binary decision variables for classification problems with binary features. Verwer & Zhang (2019) propose a model that works for numerical features with the same order of binary variables as in Günlük et al. (2021). Zhu et al. (2020) present a MIP approach for optimal decision trees with hyperplane splits (aka oblique trees). Other MIP-based approaches have been proposed in Aghaei et al. (2019; 2021). In addition to the MIP approach, SAT solvers have been explored to learn optimal decision trees (Bessiere et al., 2009; Narodytska et al., 2018; Hu et al., 2020).

[3] https://github.com/mengxianglgal/Quant-BnB

Another line of work explores pruning techniques to improve the efficiency of DP-based approaches. Angelino et al. (2017); Chen & Rudin (2018); Hu et al. (2019) solve the optimal sparse decision tree using analytical bounds on the optimal solution together with a customized bit-vector library. Lin et al. (2020) improve the efficiency of earlier approaches by using DP methods. Aglin et al. (2020) utilize DP to compute better dual bounds during BnB search. Demirović et al. (2020) show useful computational gains by using pre-computed information from sub-trees and hash functions. McTavish et al. (2021) design smart guessing strategies to improve the performance of BnB-based approaches.

As mentioned earlier, current approaches are unable to compute optimal classification/regression trees with continuous features at scale—a problem we address in this paper. Table 1 presents a summary of the key characteristics of related existing approaches vis-a-vis our proposed method `Quant-BnB`.

2. Preliminaries and Notations

2.1. Overview of optimal decision trees

Consider a supervised learning problem with n observations $\{(x_i, y_i)\}_{i\in[n]}$, each with p features $x_i \in \mathcal{X} \subseteq \mathbb{R}^p$ and response $y_i \in \mathcal{Y}$. A decision tree recursively partitions the feature space $\mathcal{X}$ into a number of hierarchical, disjoint regions, and makes a prediction for each region. In this paper, we focus on binary decision trees (i.e., every non-terminal node splits into left and right children) with axis-aligned splits. See Breiman et al. (1984) for further details.

For a decision tree T and feature vector x, let $T(x) \in \mathcal{Y}$ denote the corresponding prediction. Given a loss function $\ell(\cdot,\cdot)$ on $\mathcal{Y}\times\mathcal{Y}$, and a family $\mathcal{T}$ of decision trees, an *optimal decision tree* is a global optimal solution to the following optimization problem:

$$\min_{T\in\mathcal{T}} \quad \sum_{i=1}^{n} \ell(y_i, T(x_i)). \tag{1}$$

For regression problems with a scalar output, we can take $\mathcal{Y} \subset \mathbb{R}$ and ℓ as the squared loss: $\ell_{SE}(y,\hat{y}) := (y-\hat{y})^2$. We also consider extensions to multivariate continuous outcomes with $y \in \mathcal{Y} \subset \mathbb{R}^m$, $m \geq 1$ and $\ell_{SE}(y,\hat{y}) := \|y-\hat{y}\|^2$. For classification problems with C classes, one can take $\mathcal{Y} = [C] := \{1, 2, ..., C\}$, and ℓ to be the 0-1 (or mis-classification) loss i.e., $\ell_{01}(y,\hat{y}) = \mathbf{1}(y \neq \hat{y})$.

2.2. Data types for the feature space

In this paper, we consider the general case where x_i contains continuous and possibly binary features. If a feature f is continuous, then $\{x_{i,f}\}_{i=1}^n$ can contain at most n different values. Our approach also applies to the setting where the

number of distinct values of f is much smaller than n. Recall that if f is a binary feature then $x_{i,f} \in \{0,1\}, i \in [n]$. As mentioned earlier, when the features are all binary, the optimal decision tree can be computed quite efficiently as shown in recent works (cf Section 1). To gather some intuition, note that, if all features are binary, computing an optimal regression tree of depth d costs $O(np^d)$ operations by exhaustive search—this can be acceptable even when n is large (e.g. $n \approx 10^4 \sim 10^5$) but d is small (e.g. $d = 2, 3$). In contrast, if all features are drawn from a continuous distribution, then an exhaustive search would cost $O(n^d p^d)$ (almost surely)—this may be computationally intractable for the same values of n, d. In this paper, our focus is to propose an efficient BnB framework for the challenging case when the features are continuous—a topic that seems to be less studied when compared to the case where all features are binary.

2.3. Notations

For a feature $f \in [p]$, let $u(f) \in [n]$ be the number of distinct elements in $\{x_{i,f}\}_{i=1}^n$—i.e., the number of unique values assumed by feature f in the training data. We let $w_1^f < w_2^f < \cdots < w_{u(f)}^f$ denote these distinct values. In addition, we set $w_0^f = -\infty$ and $w_{u(f)+1}^f = +\infty$. For any integer t with $0 \le t \le u(f)$, let $\tilde{w}_t^f := (w_t^f + w_{t+1}^f)/2$. For Problem (1), it suffices to consider candidate trees with all splitting thresholds located in the set $\{\tilde{w}_t^f\}_{f\in[p],0\le t\le u(f)}$ (Note that different splitting thresholds in the interval (w_t^f, w_{t+1}^f) give the same routing decision on the training set, so we choose the mid-point $\tilde{w}_t^f$ as a candidate threshold). As we consider axis-aligned splits, each splitting node of a tree can be described by a tuple (f, t), where $f \in [p]$ is the splitting feature, and $0 \le t \le u(f)$ is an integer indicating that the splitting threshold is $\tilde{w}_t$. We say (f, t) is the *splitting rule* of this node. For a set $\mathcal{I} \subseteq [n]$, a feature $f \in [p]$ and two integers a, b with $0 \le a \le b \le u(f)$, define

$$\mathcal{I}_{[a,b]}^f \;:=\; \{i \in \mathcal{I} \mid \tilde{w}_a^f \le x_{i,f} \le \tilde{w}_b^f\}$$

to be the set of sample indices i for which $x_{i,f}$ lies between $\tilde{w}_a^f$ and $\tilde{w}_b^f$.

For $A, B > 0$, we use the notations $A = O(B)$ or $A \lesssim B$ to denote that there exists a universal constant C such that $A \le CB$; and use the notation $A = \widetilde{O}(B)$ if $A = O(B\log(n))$, where n is the number of samples.

2.4. Preliminaries for decision tree with depth 2

The algorithms we present in this paper are capable of solving shallow (e.g., $d = 2$ or 3) optimal trees within practical runtimes. We present an in-depth description of our proposed approach `Quant-BnB` for the case $d = 2$. Section 5 presents a sketch of how it can be extended to deeper trees.

An optimal tree of depth 2 is a solution to the following problem

$$\min_{T\in\mathcal{T}_2} \;\sum_{i=1}^n \ell(y_i, T(x_i)), \tag{2}$$

where, $\mathcal{T}_2$ is the set of all decision trees with depth 2 whose splitting thresholds are in $\{\tilde{w}_t^f\}_{f\in[p],0\le t\le u(f)}$.

For a tree $T \in \mathcal{T}_2$, let $(f_O(T), t_O(T))$, $(f_L(T), t_L(T))$ and $(f_R(T), t_R(T))$ denote the splitting rules at the root node (O), the left child (L) and right child (R) of the root node respectively. Given $f_0, f_1, f_2 \in [p]$ and two integers a, b with $0 \le a \le b \le u(f_0)$, we define

$$\mathcal{T}_2(f_0, [a,b], f_1, f_2) := \Big\{T \in \mathcal{T}_2 \Big| \begin{matrix} f_O(T) = f_0,\, t_O(T) \in [a,b], \\ f_L(T) = f_1,\; f_R(T) = f_2 \end{matrix}\Big\}$$

to be the set of trees in $\mathcal{T}_2$ whose root node, left child and right child splitting features are f_0, f_1 and f_2 respectively, and the splitting threshold of the root node is between $\tilde{w}_a$ and $\tilde{w}_b$. Given $F_1, F_2 \subseteq [p]$, define

$$\mathcal{T}_2(f_0, [a,b], F_1, F_2) \;:=\; \bigcup_{f_1\in F_1, f_2\in F_2} \mathcal{T}_2(f_0, [a,b], f_1, f_2).$$

We adopt the convention that $\mathcal{T}_2(f_0, [a,b], F_1, F_2) = \emptyset$ if F_1 or F_2 is empty. Note that here a and b are integers indicating that the splitting threshold of f_0 lies in $\{\tilde{w}_a^{f_0}, \tilde{w}_{a+1}^{f_0}, , ..., \tilde{w}_b^{f_0}\}$. See the appendix for an illustrative example.

For a given subset of samples $\mathcal{I} \subseteq [n]$, define

$$L_0(\mathcal{I}) \;:=\; \min_{y\in\mathcal{Y}} \sum\nolimits_{i\in\mathcal{I}} \ell(y_i, y) \tag{3}$$

to be the loss of the best constant fit to $\{y_i\}_{i\in\mathcal{I}}$. As concrete examples, for regression problem with $\mathcal{Y} = \mathbb{R}^m$ and $\ell = \ell_{SE}$, it holds

$$L_0(\mathcal{I}) \;=\; \sum\nolimits_{i\in\mathcal{I}} \big\|y_i - (1/|\mathcal{I}|)\sum\nolimits_{j\in\mathcal{I}} y_j\big\|^2. \tag{4}$$

For classification problem with $\mathcal{Y} = [C]$ and loss $\ell = \ell_{01}$, it holds

$$L_0(\mathcal{I}) \;=\; \min_{c\in[C]} \sum\nolimits_{i\in\mathcal{I}} \mathbf{1}_{\{y_i\neq c\}}. \tag{5}$$

Note that $L_0(\mathcal{I})$ can be viewed as the minimum loss of a "depth-0" decision tree with observations in $\mathcal{I}$. Similarly, we define a function L_1 to be the best possible objective value for a depth-1 decision tree (a "stump") when using observations in $\mathcal{I}$ and a feature $f \in [p]$:

$$L_1(\mathcal{I}, f) \;:=\; \min_{0\le t\le u(f)} \big\{L_0(\mathcal{I}_{[0,t]}^f) + L_0(\mathcal{I}_{[t,u(f)]}^f)\big\}. \tag{6}$$

Note that for a given feature f and a set of indices $\mathcal{I}$, if L_0 is given by (4) or (5), then $L_1(\mathcal{I}, f)$ can be computed within

$O(|\mathcal{I}|\log|\mathcal{I}|) \leq \widetilde{O}(|\mathcal{I}|)$ operations (the log-factor is from a sorting operation).

Let us consider $f_0, f_1, f_2 \in [p]$, integers a, b with $0 \leq a \leq b \leq u(f_0)$ and $\mathcal{I} \subseteq [n]$. Extending the definitions of L_0 and L_1 above, we now consider the best objective for a depth-2 decision tree. Define

$$\begin{aligned} &L_2(\mathcal{I}, f_0, [a,b], f_1, f_2) \\ &:= \min_{a \leq t \leq b} \left\{ L_1(\mathcal{I}^{f_0}_{[0,t]}, f_1) + L_1(\mathcal{I}^{f_0}_{[t,u(f_0)]}, f_2) \right\} \end{aligned} \tag{7}$$

to be the minimum value of $\sum_{i\in\mathcal{I}} \ell(y_i, T(x_i))$ over all depth-2 trees $T \in \mathcal{T}_2(f_0, [a,b], f_1, f_2)$.

To simplify the notation above, define the parameter space

$$\begin{aligned} \Phi := \Big\{ &(f_0, [a,b], f_1, f_2) \mid f_0, f_1, f_2 \in [p], \\ &a, b \text{ are integers with } 0 \leq a \leq b \leq u(f_0) \Big\}. \end{aligned} \tag{8}$$

Then for any $\phi = (f_0, [a,b], f_1, f_2) \in \Phi$, and $\mathcal{I} \subseteq [n]$, we use the short-hand notations

$$\begin{aligned} \mathcal{T}_2(\phi) &= \mathcal{T}_2(f_0, [a,b], f_1, f_2), \\ L_2(\mathcal{I}, \phi) &= L_2(\mathcal{I}, f_0, [a,b], f_1, f_2). \end{aligned}$$

3. Upper and Lower Bounds for $L_2(\mathcal{I}, \phi)$

For any $\phi \in \Phi$ and $\mathcal{I} \subseteq [n]$, recall that $L_2(\mathcal{I}, \phi)$ is the best objective value of (2) across trees in $\mathcal{T}_2(\phi)$ with samples in $\mathcal{I}$. In this section, we discuss upper and lower bounds for $L_2(\mathcal{I}, \phi)$—the costs for computing these bounds are lower than directly computing $L_2(\mathcal{I}, \phi)$. `Quant-BnB` critically makes use of these upper and lower bounds while performing BnB (cf Section 4).

3.1. Upper bounds for $L_2(\mathcal{I}, \phi)$

Given $\phi = (f_0, [a,b], f_1, f_2) \in \Phi$ and $\mathcal{I} \subseteq [n]$, we compute an upper bound of $L_2(\mathcal{I}, \phi)$ by making use of the quantiles of the features. For an integer s with $1 \leq s \leq b - a$, we say that a set of integers $(t_0, t_1, \ldots, t_s)$ are *almost s-equi-spaced* in the interval $[a,b]$ if $t_0 = a$, $t_s = b$ and $t_j = \lfloor a + (j/s)(b-a) \rfloor$ for $1 \leq j \leq s-1$. Given such a sequence $(t_0, t_1, \ldots, t_s)$, we define

$$V_s(\mathcal{I}, \phi) := \min_{0 \leq j \leq s} \left\{ L_1(\mathcal{I}^{f_0}_{[0,t_j]}, f_1) + L_1(\mathcal{I}^{f_0}_{[t_j, u(f_0)]}, f_2) \right\}.$$

It follows that $V_s(\mathcal{I}, \phi) \geq L_2(\mathcal{I}, \phi)$ for all $\phi = (f_0, [a,b], f_1, f_2) \in \Phi$ satisfying $b - a \geq s$; and hence V_s is an upper bound to L_2. We note that quantile-based methods are commonly used as a heuristic in decision tree algorithms (e.g., XGBoost (Chen & Guestrin, 2016)). Our work differs—as discussed below, we make use of this quantile-based approach to obtain an optimal decision tree.

3.2. Lower bounds for $L_2(\mathcal{I}, \phi)$

We present some lower bounds for $L_2(\mathcal{I}, \phi)$. The lower bounds along with the upper bounds discussed earlier, form the backbone of our BnB procedure.

Lower bound 1: The first lower bound we consider is obtained by sorting the values of a feature, and dropping the values lying in an interior sub-interval. Given any $\phi = (f_0, [a,b], f_1, f_2) \in \Phi$ and $\mathcal{I} \subseteq [n]$, define

$$\begin{aligned} W_0(\mathcal{I}, \phi) &= W_0(\mathcal{I}, f_0, [a,b], f_1, f_2) \\ &:= L_1(\mathcal{I}^{f_0}_{[0,a]}, f_1) + L_1(\mathcal{I}^{f_0}_{[b,u(f_0)]}, f_2). \end{aligned} \tag{9}$$

We can interpret $W_0(\mathcal{I}, \phi)$ as follows: the samples in $\mathcal{I}^{f_0}_{[0,a]}$ are routed to the left subtree yielding a loss $L_1(\mathcal{I}^{f_0}_{[0,a]}, f_1)$; the samples in $\mathcal{I}^{f_0}_{[b,u(f_0)]}$ are routed to the right subtree with a loss $L_1(\mathcal{I}^{f_0}_{[b,u(f_0)]}, f_2)$; and the samples in $\mathcal{I}^{f_0}_{[a,b]}$ are "dropped" (i.e., do not enter any leaf node). Lemma 3.2 shows that W_0 is a lower bound for L_2 i.e., $W_0(\mathcal{I}, \phi) \leq L_2(\mathcal{I}, \phi)$ for all $\phi \in \Phi$. Furthermore, $W_0(\mathcal{I}, \phi)$ is easier to compute than $L_2(\mathcal{I}, \phi)$. Figure 1 presents a schematic diagram illustrating computation of the lower bound W_0.

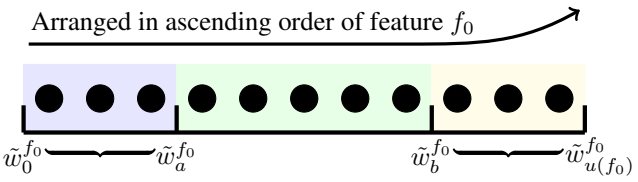


Figure 1. Figure showing sorting of the unique values in feature f_0 into intervals $[\tilde{w}_0^{f_0}, \tilde{w}_a^{f_0}]$, $[\tilde{w}_a^{f_0}, \tilde{w}_b^{f_0}]$ and $[\tilde{w}_b^{f_0}, \tilde{w}_{u(f_0)}^{f_0}]$. In the definition of W_0, samples with x_{i,f_0} between $\tilde{w}_a^{f_0}$ and $\tilde{w}_b^{f_0}$ (green part in the figure) are dropped.

Lower bound 2: The second lower bound we consider is obtained by using a subset of samples in the middle to fit another depth-2 tree. Given any $\phi = (f_0, [a,b], f_1, f_2) \in \Phi$ and $\mathcal{I} \subseteq [n]$, define

$$\begin{aligned} W_2(\mathcal{I}, \phi) &= W_2(\mathcal{I}, f_0, [a,b], f_1, f_2) \\ &:= W_0(\mathcal{I}, \phi) + L_2(\mathcal{I}^{f_0}_{[a,b]}, \phi). \end{aligned} \tag{10}$$

Note that in the definition of W_2, the samples in $\mathcal{I}^{f_0}_{[a,b]}$ are used to fit another depth-2 tree in $\mathcal{T}_2(\phi)$, and yield an additional loss term $L_2(\mathcal{I}^{f_0}_{[a,b]}, \phi)$. It can be proved that $W_2(\mathcal{I}, \phi) \leq L_2(\mathcal{I}, \phi)$ for all $\phi \in \Phi$ (see Lemma 3.2). Note that $W_2(\mathcal{I}, \phi)$ is a better lower bound than $W_0(\mathcal{I}, \phi)$, but has a higher computational cost—see Section 4.3 for details.

Lower bound 3: The third lower bound we introduce below combines the above ideas underlying the computation of W_0 and W_2. To introduce this lower bound, we need some

additional notations. Given a $\phi = (f_0, [a, b], f_1, f_2) \in \Phi$; and almost s'-equi-spaced integers $(t_0, t_1, \ldots, t'_s)$ in $[a, b]$ (for an integer $s' \leq b - a$), define

$$\begin{aligned} &\widehat{L}_2(\mathcal{I}, \phi, s') = \widehat{L}_2(\mathcal{I}, f_0, [a, b], f_1, f_2, s') \\ &:= \min_{1 \leq j \leq s'} \left\{ L_1(\mathcal{I}^{f_0}_{[0,t_{j-1}]}, f_1) + L_1(\mathcal{I}^{f_0}_{[t_j, u(f_0)]}, f_2) \right\}. \end{aligned} \tag{11}$$

Note that in the j-th term of the minimum in (11), we drop the observations in $\mathcal{I}^{f_0}_{[t_{j-1},t_j]}$. The following lemma shows that $\widehat{L}_2(\cdot, \cdot, s')$ is a lower bound of $L_2(\cdot, \cdot)$.

Lemma 3.1. *For any $\mathcal{I} \subseteq [n]$, any $\phi = (f_0, [a, b], f_1, f_2) \in \Phi$, and any $s \leq b - a$, it holds*

$$\widehat{L}_2(\mathcal{I}, \phi, s') \leq L_2(\mathcal{I}, \phi). \tag{12}$$

The proof of Lemma 3.1 is presented in the appendix. Note that the computation of $\widehat{L}_2(\mathcal{I}, \phi, s')$ is easier than the computation of $L_2(\mathcal{I}, \phi)$.

Using $\widehat{L}_2$, we can introduce the third lower bound for $L_2(\mathcal{I}, \phi)$. Given any $\phi = (f_0, [a, b], f_1, f_2) \in \Phi$ and any integer $s' \leq b - a$, define

$$\begin{aligned} &W_{1,s'}(\mathcal{I}, \phi) = W_{1,s'}(\mathcal{I}, f_0, [a, b], f_1, f_2) \\ &:= W_0(\mathcal{I}, \phi) + \widehat{L}_2(\mathcal{I}^{f_0}_{[a,b]}, \phi, s'). \end{aligned} \tag{13}$$

Note that in the definition above, the samples in $\mathcal{I}^{f_0}_{[a,b]}$ are used to compute a term $\widehat{L}_2(\mathcal{I}^{f_0}_{[a,b]}, \phi, s')$. As a result, the value $W_{1,s'}(\mathcal{I}, \phi)$ is larger than $W_0(\mathcal{I}, \phi)$ and smaller than $W_2(\mathcal{I}, \phi)$ (due to Lemma 3.1).

The following lemma justifies that W_0, $W_{1,s'}$ and W_2 are indeed lower bounds of L_2.

Lemma 3.2. *For any $\mathcal{I} \subseteq [n]$, $\phi \in \Phi$ and $s' \leq b - a$, it holds*

$$W_0(\mathcal{I}, \phi) \leq W_{1,s'}(\mathcal{I}, \phi) \leq W_2(\mathcal{I}, \phi) \leq L_2(\mathcal{I}, \phi).$$

The proof of Lemma 3.2 is presented in the appendix. Lemma 3.2 shows that W_0, $W_{1,s'}$ and W_2 are lower bounds for L_2; W_0 is weakest and W_2 is the tightest of the three bounds. See Section 4.3 for further discussions on the computational costs of these three lower bounds.

4. A Branch and Bound Method for Optimal Trees with Depth 2

In this section, we describe our algorithm `Quant-BnB` to solve problem (2) to optimality. `Quant-BnB` maintains and updates a search space represented as disjoint unions of sets of the form $\mathcal{T}_2(f_0, [a, b], F_1, F_2)$. In Section 4.1, we first present a proposition illustrating how we perform quantile-based pruning on a set of the form $\mathcal{T}_2(f_0, [a, b], F_1, F_2)$. In Section 4.2 we present the overall framework of `Quant-BnB`. In Section 4.3 we discuss the computational cost of the algorithm.

4.1. A quantile-based pruning procedure

Suppose we are given $f_0 \in [p]$, integers a, b with $0 \leq a \leq b \leq u(f_0)$ and $F_1, F_2 \subseteq [p]$. We focus on the subset of trees $\mathcal{T}_2(f_0, [a, b], F_1, F_2)$, and discuss a quantile-based method to replace this collection by a smaller subset containing an optimal solution to (2).

Let $(t_0, ..., t_s)$ be almost s-equi-spaced in $[a, b]$. The following proposition states a basic strategy for pruning.

Proposition 4.1. *Let W be a function on $2^{[n]} \times \Phi$ with $W(\mathcal{I}, \phi) \leq L_2(\mathcal{I}, \phi)$ for all $\phi \in \Phi$ and $\mathcal{I} \subseteq [n]$; let U be an upper bound of the optimal value of problem (2). For each $j \in [s]$, define*

$$F_{1,j} := \left\{ f_1 \in F_1 \,\middle|\, \min_{f_2 \in F_2} W([n], \phi^j_{f_1,f_2}) \leq U \right\}, \tag{14}$$

$$F_{2,j} := \left\{ f_2 \in F_2 \,\middle|\, \min_{f_1 \in F_1} W([n], \phi^j_{f_1,f_2}) \leq U \right\}, \tag{15}$$

where $\phi^j_{f_1,f_2} := (f_0, [t_{j-1}, t_j], f_1, f_2)$. Then any optimal solution of (2) is not in

$$\mathcal{T}_2(f_0, [a, b], F_1, F_2) \setminus \bigcup_{j=1}^{s} \mathcal{T}_2(f_0, [t_{j-1}, t_j], F_{1,j}, F_{2,j}). \tag{16}$$

The proof of Proposition 4.1 is presented in the appendix. Possible choices of the lower bound $W(\mathcal{I}, \phi)$ in Proposition 4.1 are $W_0(\mathcal{I}, \phi)$, $W_{1,s'}(\mathcal{I}, \phi)$ and $W_2(\mathcal{I}, \phi)$, as designed in Section 3.2. If the assumptions of Proposition 4.1 hold, we can replace the search space $\mathcal{T}_2(f_0, [a, b], F_1, F_2)$ by a smaller space $\cup_{j=1}^{s} \mathcal{T}_2(f_0, [t_{j-1}, t_j], F_{1,j}, F_{2,j})$, or equivalently, prune all the feasible solutions in (16). Note that it is possible that for some $j \in [s]$, the set $F_{1,j}$ or $F_{2,j}$ is empty, and hence the set $\mathcal{T}_2(f_0, [t_{j-1}, t_j], F_{1,j}, F_{2,j})$ is also empty. In that case, we prune the trees in $\mathcal{T}_2(f_0, [a, b], F_1, F_2)$ with splitting thresholds lying in $[t_{j-1}, t_j]$.

4.2. `Quant-BnB` framework

We discuss the overall methodology of `Quant-BnB` to solve (2). The proposed algorithm maintains and updates a set $\texttt{AL}^{(k)}$ (short for "alive") over iterations $k \geq 0$. $\texttt{AL}^{(k)}$ contains tuples of the form

$$(f_0, [a, b], F_1, F_2),$$

where $f_0 \in [p]$; a and b are integers with $0 \leq a \leq b \leq u(f_0)$; and $F_1, F_2 \subseteq [p]$. Initially (i.e., at $k = 0$), all the trees in $\mathcal{T}_2$ are "alive", so we set

$$\texttt{AL}^{(0)} = \cup_{f_0=1}^{p} \left\{ (f_0, [0, u(f_0)], [p], [p]) \right\}. \tag{17}$$

Intuitively speaking, as the iterations progress, we reduce the size of $\texttt{AL}^{(k)}$, by removing tuples $(f_0, [a, b], F_1, F_2)$ that do not contain optimal solutions to (2). The algorithm also maintains and updates the best objective value that it has found so far, denoted by U. Initially, we set U to be the value at any feasible solution of (2).

At every iteration $k \geq 1$, to update $\texttt{AL}^{(k)}$ from $\texttt{AL}^{(k-1)}$, we first set $\texttt{AL}^{(k)} = \emptyset$. The algorithm then checks all elements in $\texttt{AL}^{(k-1)}$. For an element $(f_0, [a, b], F_1, F_2)$ in $\texttt{AL}^{(k-1)}$, if $b-a$ is less than a pre-specified integer s (i.e., the number of trees in the space is sufficiently small), our algorithm performs an exhaustive search to examine all candidate trees in the space $\mathcal{T}_2(f_0, [a, b], F_1, F_2)$. Otherwise, the algorithm conducts the following 2 steps. Let $(t_0, ..., t_s)$ be an almost s-equi-spaced sequence in $[a, b]$.

- (Step1: Update upper bound) Compute

$$U' = \min_{f_1 \in F_1, f_2 \in F_2} \big\{V_s([n], f_0, [a, b], f_1, f_2)\big\}.$$

  If $U' < U$, update $U \leftarrow U'$, and update the corresponding best tree.

- (Step2: Compute lower bound and prune) For a function W on $2^{[n]} \times \Phi$ satisfying $W(\mathcal{I}, \phi) \leq L_2(\mathcal{I}, \phi)$ for all $\phi \in \Phi$ and $\mathcal{I} \subseteq [n]$, compute sets $F_{1,j}$ and $F_{2,j}$ as in (14) and (15) (for all $j \in [s]$), and update

$$\texttt{AL}^{(k)} = \texttt{AL}^{(k)} \bigcup \Big(\cup_{j=1}^{s} \{(f_0, [t_{j-1}, t_j], F_{1,j}, F_{2,j})\}\Big).$$

Note that in Step 2 above, we need to compute values of $W([n], \phi^j_{f_1,f_2})$ as in (14) and (15)–these are lower bounds of $L_2([n], \phi^j_{f_1,f_2})$. Function W can be taken as W_0, $W_{1,s'}$ or W_2, as introduced in Section 3.2. At the end of Step 2, the set $\cup_{j=1}^{s} \{(f_0, [t_{j-1}, t_j], F_{1,j}, F_{2,j})\}$ is added into $\texttt{AL}^{(k)}$; this set replaces the tuple $(f_0, [a, b], F_1, F_2)$ in $\texttt{AL}^{(k-1)}$. In other words, all the trees that lie in $\mathcal{T}_2(f_0, [a, b], F_1, F_2)$ but not in $\cup_{j=1}^{s} \mathcal{T}_2(f_0, [t_{j-1}, t_j], F_{1,j}, F_{2,j})$ are pruned.

The main steps of the algorithm above are summarized in Algorithm 1. For illustration, a single iteration of `Quant-BnB` on a toy example is provided in appendix.

4.3. Correctness of `Quant-BnB`, computational cost

Theorem 4.2 (see Appendix for proof) establishes that `Quant-BnB` converges to the global optimum of (2).

Theorem 4.2. *Algorithm 1 terminates in at most $\lceil log_s(n) \rceil$ iterations and yields an optimal solution of (2).*

Computational cost: We discuss the computational cost of Steps 1 & 2 for a given $(f_0, [a, b], F_1, F_2)$. To simplify the discussion, we consider the case when the number of unique values of feature f_0 i.e., $u(f_0)$ is n.

Algorithm 1 `Quant-BnB` for depth-2 decision trees

Input: data $\{(x_i, y_i)\}_{i=1}^n$, an integer $s \geq 2$, an initial upper bound U and a feasible solution $\hat{T}$.
Initialize $\texttt{AL}^{(0)}$ as in (17), and set $k = 1$.
repeat
  Set $\texttt{AL}^{(k)} = \emptyset$.
  for each $(f_0, [a, b], F_1, F_2)$ in $\texttt{AL}^{(k-1)}$ **do**
    if $b - a \leq s$ **then**
      Use exhaustive search to get the bound $U' = \min_{f_1 \in F_1, f_2 \in F_2} L_2([n], f_0, [a, b], f_1, f_2)$.
      Update $U = \min\{U, U'\}$ and accordingly, the current best solution $\hat{T}$.
      continue
    end if
    Let $(t_0, t_1, \ldots, t_s)$ be almost s-equi-spaced in $[a, b]$.
    Perform Steps 1 and 2.
  end for
  Update $k \leftarrow k + 1$.
until $\texttt{AL}^{(k)}$ is empty
Output: The optimal decision tree $\hat{T}$ and corresponding objective value U.

Note that in Algorithm 1, a function W satisfying $W(\mathcal{I}, \phi) \leq L_2(\mathcal{I}, \phi)$ for all $\phi \in \Phi$ and $\mathcal{I} \subseteq [n]$ is needed. Candidates of the lower bound W have been discussed in Section 3.2. Different choices of W have different computational costs. The simplest choice is to set $W = L_2$ directly, in which case Step 2 of the algorithm reduces to an exhaustive search over $\mathcal{T}_2(f_0, [a, b], F_1, F_2)$, which is expensive. By Lemma 3.2, it is also possible to take W to be W_0, $W_{1,s'}$ (for some proper integer s') or W_2. We compare the computational cost of Step 1 – Step 2 under these different choices, as shown below.

Lemma 4.3. *Suppose L_0 is given by (4) or (5). For a given $(f_0, [a, b], F_1, F_2)$, denote $\tilde{p} := |F_1| + |F_2|$. Suppose $u(f_0) = n$. Let s, s' be positive integers with $s' \cdot s \leq b - a$. The computational cost of Steps 1-2 for different choices of the lower-bound function W are as follows:*
(1) If $W = W_0$, the cost is bounded by $\widetilde{O}(n\tilde{p}s)$.
(2) If $W = W_{1,s'}$, the cost is bounded by $\widetilde{O}(n\tilde{p}s + \tilde{p}s'(b-a))$.
(3) If $W = W_2$, the cost is bounded by $\widetilde{O}(n\tilde{p}s + \frac{(b-a)^2\tilde{p}}{s})$.
(4) If $W = L_2$, the cost is bounded by $\widetilde{O}(n\tilde{p}(b-a))$.

Note that the assumption $s \cdot s' \leq b - a$ is necessary to make sure the equi-spaced sequences are well-defined. By this assumption, we have $n\tilde{p}s \leq n\tilde{p}s + \tilde{p}s'(b-a) \leq n\tilde{p}s + (b-a)^2\tilde{p}/s \lesssim n\tilde{p}(b-a)$. Lemma 4.3 implies that

$$W_0 \prec W_{1,s'} \prec W_2 \prec L_2, \tag{18}$$

where the notation "$\bar{W} \prec \tilde{W}$" means that the cost (of Steps 1 & 2) in using $W = \bar{W}$ is bounded by a constant multiple of

the cost of using $W = \tilde{W}$. On the other hand, by Lemma 3.2, it is known that

$$W_0(\mathcal{I}, \phi) \le W_{1,s'}(\mathcal{I}, \phi) \le W_2(\mathcal{I}, \phi) \le L_2(\mathcal{I}, \phi) \quad (19)$$

for all $\phi \in \Phi$. Therefore, among these four choices of W, there is a tradeoff: the choice with lower computational time for Step 1 – Step 2 will produce a less tight lower bound and may result in a case where fewer trees are pruned in this iteration. Empirically, we find that among these four choices, choosing $W = W_{1,s'}$ (for a proper s') has the best performance in most cases (see Section 6.1 for a comparison of these choices). A choice of s' that appears to work well in practice is $s' \approx \frac{ns}{b-a}$, in which case the cost in Lemma 4.3 (2) reduces to $\widetilde{O}(n\tilde{p}s)$. With this choice of s', the cost of Step 1–2 using $W = W_{1,s'}$ is the same (up to a constant multiple) as the cost of choosing $W = W_0$, but the former always provides a better lower bound.

We note that when choosing $W = W_2$ or $W = L_2$, the cost of Algorithm 1 is not linear in n. Indeed, initially ($k = 0$), for any $(f_0, [a, b], F_1, F_2)$ in $\texttt{AL}^{(0)}$, it holds $b - a = n$, and $\tilde{p} = 2p$. So by Lemma 4.3 (4) the cost of Steps 1–2 with $W = L_2$ is $\widetilde{O}(n^2\tilde{p})$; and with $W = W_2$ the cost is $\widetilde{O}(nps + (n^2p/s)) \ge \widetilde{O}(n^{3/2}p)$.

5. Extension to Deeper Trees

Algorithm 1 can be generalized for the computation of optimal trees with a fixed depth d ($d \ge 3$). We briefly discuss the case when $d = 3$. To compute an optimal tree of depth 3, `Quant-BnB` maintains and updates a set `AL3` that contains tuples of the form

$$(f_0, [a, b], \Phi_1, \Phi_2), \quad (20)$$

where $f_0 \in [p]$, $0 \le a \le b \le u(f_0)$, and $\Phi_1, \Phi_2 \subseteq \Phi$. Recall that Φ is defined in (8), which contains tuples corresponding to subsets of depth-2 trees. Each tuple (20) corresponds to a search space in which the elements meet the following conditions: the root node (f_0, t_0) satisfies $t_0 \in [a, b]$; also, the left and right branch of the root node, which are decision trees of depth 2, are in $\mathcal{T}_2(\phi_1)$ and $\mathcal{T}_2(\phi_2)$ for some $\phi_1 \in \Phi_1$ and $\phi_2 \in \Phi_2$ respectively.

To shrink the search space corresponding to (20), we set up an almost equi-spaced sequence of integers and work with lower bounds for each smaller search space. Due space limits, we present the details of the algorithm and related discussions in Section C.

For the case $d \ge 4$, similar recursion can be applied to design BnB algorithms, but the computational cost increases especially when p (# of features) is large. Therefore, we recommend using our procedure for fitting optimal decision trees with $d \le 3$.

So far, we only consider perfect binary trees (i.e., depth-d trees that has exactly $2^d - 1$ branch nodes). In practice, it is preferable to optimize over non-perfect trees to enhance generalization capability, especially for deeper trees. Note that we can modify `Quant-BnB` to handle non-perfect trees by considering two cases for each node–it can be a branch node or a leaf–when calculating upper and lower bounds. This modification will not result in additional computation costs.

6. Numerical Experiments

In this section, we study the performance of `Quant-BnB` in terms of runtime and prediction accuracy. In particular we study: (i) differences in the efficiency of `Quant-BnB` using various methods to calculate the lower bound of $L_2(\mathcal{I}, \phi)$ (ii) computational cost of optimal trees (depths 2 and 3) on classification tasks compared to state-of-the-art algorithms (iii) out-of-sample accuracy compared to heuristic methods. We present details of experimental setup and results on regression tasks in appendix Section D.

Datasets and Computing Environment: We collect 16 classification (binary and multi-class) and 11 regression datasets from UCI Machine Learning Repository (Dua & Graff, 2017). All experiments are carried out on MIT's Engaging cluster on Intel Xeon 2.30GHz machine, with a single CPU core and 25GB of RAM. Our algorithm implementation can be found on GitHub[4].

6.1. Comparison of different lower bounds

Recall that `Quant-BnB` requires a lower-bound function W such that $W(\mathcal{I}, \phi) \le L_2(\mathcal{I}, \phi)$ for all $\phi \in \Phi$ and $\mathcal{I} \subseteq [n]$. In addition, the efficiency of the algorithm depends largely on the choice of W. We have developed 4 possible lower bounds—W_0, $W_{1,s'}$, W_2 and L_2, and theoretically studied the quality and computational cost of them (see (18) and (19)). In this experiment, we figure out their practical performance.

We set the parameter s in Algorithm 1 to be 3, and the parameter s' is dynamically chosen as $\lfloor \frac{0.6ns}{b-a} \rfloor$ for tuple $\phi = (f_0, [a, b], f_1, f_2)$. As discussed in Section 4.3, the computational cost of W_0 and $W_{1,s'}$ is linear w.r.t n under such setting, while computing W_2 and L_2 in the first iteration of `Quant-BnB` costs $O(n^2)$.

We implement `Quant-BnB` with the lower-bound function W in Step 1-2 chosen as W_0, $W_{1,s'}$, W_2 and L_2, respectively. Table 3 displays the computation time of these four choices on UCI datasets with number of data points $n \ge 10^4$. Although W_2 produces a tighter lower bound compared to

[4] https://github.com/mengxianglgal/Quant-BnB

W_0 and $W_{1,s'}$ (which helps `Quant-BnB` prune more trees in every iteration), taking $W = W_2$ still has a bad performance due to its expensive computational cost. In contrast, proposed lower bounds W_0 and $W_{1,s'}$ result in significant speedups compared to computing L_2 directly. In the following experiments, we always choose $W = W_{1,s'}$ to reduce computational cost.

Name	(n,p)	W_0	$W_{1,s'}$	W_2	L_2
avila	(10432,10)	5.1	4.5	-	519
bean	(10888,16)	3.0	3.4	-	-
eeg	(11984,14)	2.9	2.9	-	47
htru	(14318,8)	1.4	1.3	244	515
magic	(15216,10)	1.2	1.0	-	-
skin	(196045,3)	2.0	2.1	-	31
casp	(36584,9)	6.7	4.2	-	-
energy	(15788,28)	18	14	-	-
gas	(29386,10)	1.5	1.5	-	444
news	(31715,59)	301	349	-	-
query2	(159874,4)	15	9.8	-	-

Table 3. `Quant-BnB` with four different methods for lower bound computation on depth-2 trees. For each dataset, the number of observations and the number of features are provided. Each entry denotes running time in seconds. Symbol '-' refers to time out (10min).

6.2. Comparison with state-of-the-art optimal methods

We compare our algorithm with the recently proposed methods for solving optimal classification trees: BinOCT (Verwer & Zhang, 2019), MurTree (Demirović et al., 2020) and DL8.5 (Aglin et al., 2020). We also tested other competing algorithms, but they all took substantially longer time to deliver optimal trees—see Appendix for details. Since MurTree and DL8.5 apply to datasets with binary features, we adopt the equivalent-conversion pre-processing used in Lin et al. (2020) by encoding each continuous feature f to a set of $u(f) - 1$ binary features, using all possible thresholds.

The computation time for learning optimal shallow trees (depth $= 2, 3$) on classification tasks is presented in Table 2. `Quant-BnB` can solve depth-2 trees in a few seconds–a speedup of several orders of magnitude compared to other methods. When the depth is 3, `Quant-BnB` still outperforms competing algorithms by a large margin on 13 of 16 datasets. Although being highly effective on datasets with purely binary features, MurTree and DL8.5 can be expensive to deliver optimal trees on datasets with continuous features. This is perhaps due to the increase in number of features while converting the continuous features to binary features. On a related note, it is worth pointing out that one may use approximate methods to convert continuous to binary features Demirović et al. (2020)—however, such approximate schemes may result in a lossy compression of the training data as shown in our experiments in the Appendix. Note also that we do not use any compression of features for `Quant-BnB`. The numerical results illustrate the effectiveness of `Quant-BnB` in solving shallow optimal trees on large datasets with continuous features.

Dataset Name	(n,p)	depth=2				depth=3			
		Quant-BnB	BinOCT	MurTree	DL8.5	Quant-BnB	BinOCT	MurTree	DL8.5
avila	(10430,10)	**4.5**	(1.3%)	OoM	3278	**4188**	OoM	OoM	OoM
bank	(1097,4)	**<0.1**	2963	8.4	4.6	**4.4**	(100%)	142	-
bean	(10888,16)	**3.4**	(0%)	OoM	OoM	**1014**	OoM	OoM	OoM
bidding	(5056,9)	**0.2**	(1.1%)	345	72	**30**	(154%)	6252	OoM
eeg	(11984,14)	**2.9**	(6.3%)	288	34	4042	(13%)	**1783**	OoM
fault	(1552,27)	**1.6**	(9.1%)	530	271	-	(34%)	-	OoM
htru	(14318,8)	**1.3**	(7.7%)	OoM	OoM	**10303**	OoM	OoM	OoM
magic	(15216,10)	**1.0**	(2.6%)	OoM	OoM	**1090**	(14%)	OoM	OoM
occupancy	(8143,5)	**0.3**	(2.3%)	193	33	**106**	(28%)	1692	OoM
page	(4378,10)	**0.4**	(0%)	155	84	**471**	(35%)	-	OoM
raisin	(720,7)	**0.1**	9590	13	6.2	**167**	(6.6%)	432	-
rice	(3048,7)	**0.4**	(3.0%)	591	267	**1340**	(11%)	-	OoM
room	(8103,16)	**1.0**	(25%)	18	14	**180**	(239%)	269	-
segment	(1848,18)	**1.1**	(0.5%)	389	213	**153**	(250%)	-	OoM
skin	(196045,3)	**2.3**	(28%)	37	16	350	(9.9%)	**112**	-
wilt	(4339,5)	**0.2**	(0%)	653	314	**67**	(56%)	-	OoM

Table 2. Comparison of `Quant-BnB` aginst BinOCT, MurTree and DL8.5 on 16 classification datasets. For each dataset, the number of observations and the number of features are provided. Each entry denotes running time in seconds. - refers to time out (4h), OoM refers to out of memory (25GB). If BinOCT times out, we display the relative difference $(L_B - L_Q)/L_Q$ as a percentage instead, where L_B and L_Q are the training errors of BinOCT and `Quant-BnB`, respectively.

6.3. Comparison with heuristic methods

We study the test-accuracy of optimal decision trees. Earlier work (Verwer & Zhang, 2019; Demirović et al., 2020) in classification with binary features suggest that optimal trees can lead to better test-performance compared to heuristics. We explore if similar empirical findings hold true for the tasks we consider herein. We compare our approach with the well-known algorithm CART (Breiman et al., 1984) and Tree Alternating Optimization (TAO) (Carreira-Perpinán & Tavallali, 2018)—both based on heuristics. Both CART, TAO consider the same models as `Quant-BnB`, namely, axis-aligned trees with depth 2 or 3.

We compare the test error on a collection of 27 datasets: 16 classification and 11 regression tasks (see Appendix for details). Since the range of loss function of each test set varies, we study the relative loss $(L_C^{te} - L_Q^{te})/L_Q^{te}$, where L_Q^{te} is test error of `Quant-BnB`, and L_C^{te} is the test error of the competing algorithm (here, C is CART or TAO). The results are summarized in Fig 2. We observe from the figure that `Quant-BnB` obtains depth-2 trees with lower test error in more than 66% datasets. When the depth is 3, `Quant-BnB` leads to better generalization in most of cases compared to CART and TAO. The prediction performance of TAO is slightly better than CART, at the expense of higher computational cost. The results indicate that optimal trees delivered by `Quant-BnB` offer an edge compared to heuristic methods, especially for deeper trees.

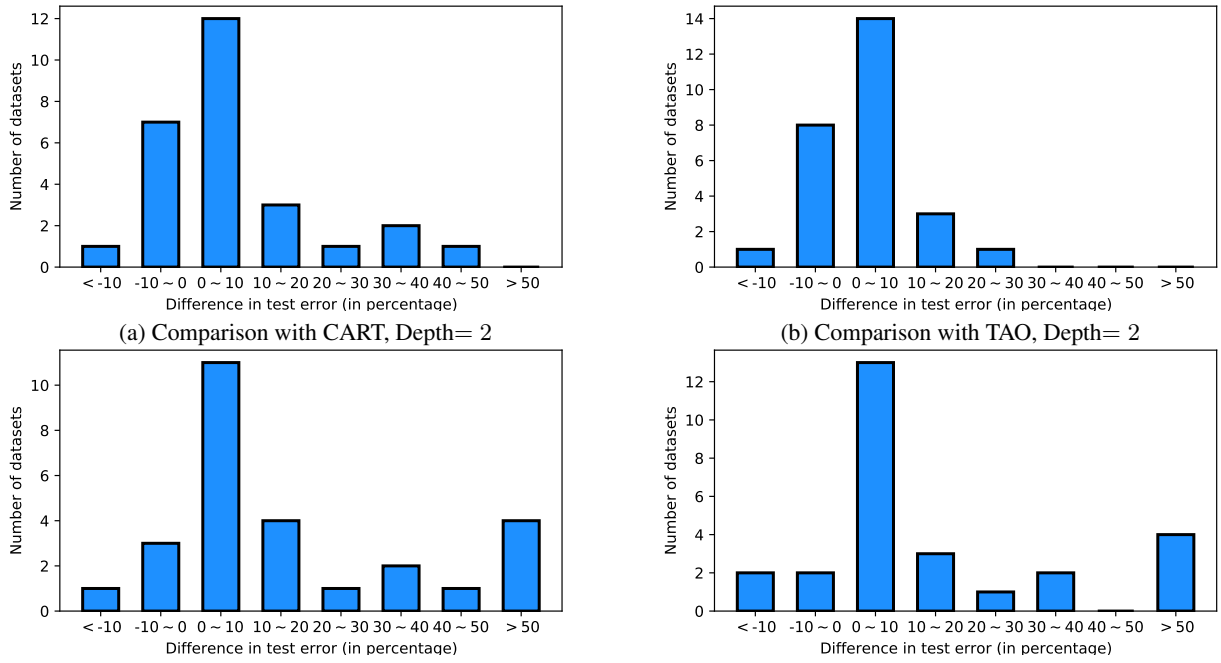
7. Conclusions and Discussions

We present a novel BnB framework for optimal decision trees that applies to both regression and classification problems with continuous features. This extend the scope of optimal procedures in the literature that have been developed for classification problems with binary features. Our approach is based on partitioning the feature values based on quantiles and using them to generate upper and lower bounds. We discuss convergence guarantees of our procedure. Numerical experiments suggest the efficiency of our approach for shallow decision trees.

Although a single optimal shallow tree appears to be somewhat restrictive in terms of prediction, it can be useful for interpretability (it can be difficult to interpret trees with depth much larger than 3). Additionally, a heuristic procedure (e.g., CART) may require a larger depth to achieve the same training/test error as an optimal tree with $d = 3$. To improve prediction performance of a single shallow tree, one can use an ensemble (e.g., random forest, Boosting) of shallow trees.

Acknowledgements

We thank the reviewers for their comments that resulted in improvements in the paper. This research is supported in part by grants from the Office of Naval Research (N00014-21-1-2841) and Liberty Mutual Insurance.



(a) Comparison with CART, Depth= 2

(b) Comparison with TAO, Depth= 2

(c) Comparison with CART, Depth= 3

(d) Comparison with TAO, Depth= 3

Figure 2. Performance comparison of `Quant-BnB` against CART and TAO on 27 datasets. L_C^{te} is test error of algorithm C (i.e., CART or TAO) and L_Q^{te} is test error of `Quant-BnB`. We summarize the relative difference $(L_C^{te} - L_Q^{te})/L_Q^{te}$-values between a heuristic method (C) and optimal decision trees delivered by `Quant-BnB` (Q), shown in percentages using bar charts. A positive value of the x-axis means `Quant-BnB` performs better than competing methods.

References

Aghaei, S., Azizi, M. J., and Vayanos, P. Learning optimal and fair decision trees for non-discriminative decision-making. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 1418–1426, 2019.

Aghaei, S., Gómez, A., and Vayanos, P. Strong optimal classification trees. *arXiv preprint arXiv:2103.15965*, 2021.

Aglin, G., Nijssen, S., and Schaus, P. Learning optimal decision trees using caching branch-and-bound search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 3146–3153, 2020.

Angelino, E., Larus-Stone, N., Alabi, D., Seltzer, M., and Rudin, C. Learning certifiably optimal rule lists for categorical data. *arXiv preprint arXiv:1704.01701*, 2017.

Bennett, K. P. and Blue, J. A. Optimal decision trees. *Rensselaer Polytechnic Institute Math Report*, 214:24, 1996.

Bertsimas, D. and Dunn, J. Optimal classification trees. *Machine Learning*, 106(7):1039–1082, 2017.

Bertsimas, D. and Dunn, J. *Machine learning under a modern optimization lens*. Dynamic Ideas LLC, 2019.

Bessiere, C., Hebrard, E., and O'Sullivan, B. Minimising decision tree size as combinatorial optimisation. In *International Conference on Principles and Practice of Constraint Programming*, pp. 173–187. Springer, 2009.

Breiman, L., Friedman, J., Olshen, R., and Stone, C. Classification and regression trees. 1984.

Carreira-Perpinán, M. A. and Tavallali, P. a. Alternating optimization of decision trees, with application to learning sparse oblique trees. *Advances in Neural Information Processing Systems*, 31:1211–1221, 2018.

Chen, C. and Rudin, C. An optimization approach to learning falling rule lists. In *International Conference on Artificial Intelligence and Statistics*, pp. 604–612. PMLR, 2018.

Chen, T. and Guestrin, C. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

Demirović, E., Lukina, A., Hebrard, E., Chan, J., Bailey, J., Leckie, C., Ramamohanarao, K., and Stuckey, P. J. Murtree: optimal classification trees via dynamic programming and search. *arXiv preprint arXiv:2007.12652*, 2020.

Dobkin, D., Fulton, T., Gunopulos, D., Kasif, S., and Salzberg, S. Induction of shallow decision trees. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 1997.

Dua, D. and Graff, C. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Farhangfar, A., Greiner, R., and Zinkevich, M. A fast way to produce near-optimal fixed-depth decision trees. In *Proceedings of the 10th international symposium on artificial intelligence and mathematics (ISAIM-2008)*, 2008.

Günlük, O., Kalagnanam, J., Li, M., Menickelly, M., and Scheinberg, K. Optimal decision trees for categorical data via integer programming. *Journal of Global Optimization*, pp. 1–28, 2021.

Hu, H., Siala, M., Hébrard, E., and Huguet, M.-J. Learning optimal decision trees with maxsat and its integration in adaboost. In *IJCAI-PRICAI 2020, 29th International Joint Conference on Artificial Intelligence and the 17th Pacific Rim International Conference on Artificial Intelligence*, 2020.

Hu, X., Rudin, C., and Seltzer, M. Optimal sparse decision trees. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Laurent, H. and Rivest, R. L. Constructing optimal binary decision trees is np-complete. *Information processing letters*, 5(1):15–17, 1976.

Lin, J., Zhong, C., Hu, D., Rudin, C., and Seltzer, M. Generalized and scalable optimal sparse decision trees. In *International Conference on Machine Learning*, pp. 6150–6160. PMLR, 2020.

McTavish, H., Zhong, C., Achermann, R., Karimalis, I., Chen, J., Rudin, C., and Seltzer, M. How smart guessing strategies can yield massive scalability improvements for sparse decision tree optimization. *arXiv preprint arXiv:2112.00798*, 2021.

Narodytska, N., Ignatiev, A., Pereira, F., Marques-Silva, J., and RAS, I. Learning optimal decision trees with sat. In *IJCAI*, pp. 1362–1368, 2018.

Nijssen, S. and Fromont, E. Mining optimal decision trees from itemset lattices. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 530–539, 2007.

Nijssen, S. and Fromont, E. Optimal constraint-based decision tree induction from itemset lattices. *Data Mining and Knowledge Discovery*, 21(1):9–51, 2010.

Quinlan, J. R. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

Verwer, S. and Zhang, Y. Learning optimal classification trees using a binary linear program formulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 1625–1632, 2019.

Zhu, H., Murali, P., Phan, D., Nguyen, L., and Kalagnanam, J. A scalable mip-based method for learning optimal multivariate decision trees. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1771–1781. Curran Associates, Inc., 2020.

Appendix

A. Examples

A.1. An example of the space $\mathcal{T}_2(f_0, [a, b], F_1, F_2)$

We consider a classification dataset with $n = 6$, $p = 3$. The feature vectors and labels are provided in Table 4.

Data	x_1	x_2	x_3	x_4	x_5	x_6
Feature 1	1	2	3	3	4	5
Feature 2	0	1	2	3	4	5
Feature 3	0	0	3	3	5	5
Label	1	2	1	2	1	2

Table 4. An classification example with 6 samples. Each has 3 features and a label in $\{1, 2\}$.

.

Then we have $u(1) = 5$, and

$$w_0^1 = -\infty,\ \ w_1^1 = 1,\ \ w_2^1 = 2,\ \ w_3^1 = 3,\ \ w_4^1 = 4,\ \ w_5^1 = 5,\ \ w_6^1 = \infty, \tag{21}$$

$$\tilde{w}_0^1 = -\infty,\ \ \tilde{w}_1^1 = 1.5,\ \ \tilde{w}_2^1 = 2.5,\ \ \tilde{w}_3^1 = 3.5,\ \ \tilde{w}_4^1 = 4.5,\ \ \tilde{w}_5^1 = \infty. \tag{22}$$

As an example, the set $\mathcal{T}_2(1, [1, 4], \{2, 3\}, \{2, 3\})$ contains all trees whose splitting features at the root node are 1; splitting thresholds at the root node are in $\{1.5,\ 2.5,\ 3.5,\ 4.5\}$; splitting features at the left child and the right child are in $\{2, 3\}$.

A.2. An example of a single iteration of `Quant-BnB`

We follow the assumption on data given in the previous section. Now suppose at some iteration k, the set $\texttt{AL}^{(k-1)}$ contains a single tuple $(1, [1, 4], \{2, 3\}, \{2, 3\})$. The current upper bound U equals to 2, and the parameter s in Algorithm 1 is set to be 2. In addition, we choose $W_0(\mathcal{I}, \phi)$ defined in Eq.(9) as the lower bound required in Proposition 4.1.

To construct $\texttt{AL}^{(k)}$, `Quant-BnB` checks the tuple $(1, [1, 4], \{2, 3\}, \{2, 3\})$. Since $s = 2 \leq 4 - 1$, the algorithm computes $(t_0, t_1, t_2) = (1, 2, 4)$ being almost 2-equi-spaced in $[1, 4]$ and conducts the following 2 steps.

- (Step1: Update upper bound) `Quant-BnB` computes U' by

$$\begin{aligned} U' &= \min_{f_1 \in \{2,3\}, f_2 \in \{2,3\}} \left\{ V_2([6], 1, [1, 4], f_1, f_2) \right\} \\ &= \min_{f_1 \in \{2,3\}, f_2 \in \{2,3\}} \Bigg\{ \min \Big\{ L_1(\{1\}, f_1) + L_1(\{2, 3, 4, 5, 6\}, f_2), \\ &\qquad\qquad L_1(\{1, 2\}, f_1) + L_1(\{3, 4, 5, 6\}, f_2),\ \ L_1(\{1, 2, 3, 4, 5\}, f_1) + L_1(\{6\}, f_2) \Big\} \Bigg\} \\ &= 1. \end{aligned}$$

  Since $U' < U$, `Quant-BnB` then updates $U = U' = 1$.

- (Step2: Compute lower bound and prune) `Quant-BnB` computes $W_0([n], \phi^j_{f_1, f_2})$ for any $j \in \{1, 2\}$, $f_1,\ f_2 \in \{2, 3\}$. The results are (computation details are omitted for simplicity)

$$\begin{aligned} &W_0([n], \phi^1_{2,2}) = 1,\ W_0([n], \phi^1_{2,3}) = 2,\ W_0([n], \phi^1_{3,2}) = 1,\ W_0([n], \phi^1_{3,3}) = 2, \\ &W_0([n], \phi^2_{2,2}) = 0,\ W_0([n], \phi^2_{2,3}) = 1,\ W_0([n], \phi^2_{3,2}) = 0,\ W_0([n], \phi^2_{3,3}) = 1. \end{aligned}$$

The algorithm then computes sets $F_{1,j}$ and $F_{2,j}$ according to (14) and (15) as

$$\begin{aligned}
F_{1,1} &:= \big\{f_1 \in F_1 \;\big|\; \min_{f_2 \in F_2} W([n], \phi^1_{f_1,f_2}) \le U\big\} = \{2,3\},\\
F_{1,2} &:= \big\{f_1 \in F_1 \;\big|\; \min_{f_2 \in F_2} W([n], \phi^2_{f_1,f_2}) \le U\big\} = \{2,3\},\\
F_{2,1} &:= \big\{f_2 \in F_2 \;\big|\; \min_{f_1 \in F_1} W([n], \phi^1_{f_1,f_2}) \le U\big\} = \{2\},\\
F_{2,2} &:= \big\{f_2 \in F_2 \;\big|\; \min_{f_1 \in F_1} W([n], \phi^2_{f_1,f_2}) \le U\big\} = \{2,3\}.
\end{aligned}$$

Finally, `Quant-BnB` updates

$$\texttt{AL}^{(k)} = \{(1,[1,2],F_{1,1},F_{2,1})\} \cup \{(1,[2,4],F_{1,2},F_{2,2})\}.$$

B. Appendix for proofs

B.1. Auxiliary results

We first prove a basic equality for L_0.

Lemma B.1. *For any disjoint sets $\mathcal{I}, \mathcal{J} \subseteq [n]$, it holds*

$$L_0(\mathcal{I} \cup \mathcal{J}) \ge L_0(\mathcal{I}) + L_0(\mathcal{J}). \tag{23}$$

Proof. Note that

$$\begin{aligned}
L_0(\mathcal{I} \cup \mathcal{J}) \;&=\; \min_{y \in \mathcal{Y}} \sum_{i \in \mathcal{I} \cup \mathcal{J}} \ell(y_i, y) \;=\; \min_{y \in \mathcal{Y}} \Big\{ \sum_{i \in \mathcal{I}} \ell(y_i, y) + \sum_{j \in \mathcal{J}} \ell(y_j, y) \Big\} \\
&\ge\; \min_{y \in \mathcal{Y}} \Big\{ \sum_{i \in \mathcal{I}} \ell(y_i, y) \Big\} + \min_{y \in \mathcal{Y}} \Big\{ \sum_{j \in \mathcal{J}} \ell(y_j, y) \Big\} \;=\; L_0(\mathcal{I}) + L_0(\mathcal{J}).
\end{aligned} \tag{24}$$

□

Using the equality above, we prove a useful inequality for L_1 presented below.

Lemma B.2. *For any disjoint sets $\mathcal{I}, \mathcal{J} \subseteq [n]$ and any $f \in [p]$, it holds*

$$L_1(\mathcal{I} \cup \mathcal{J}, f) \ge L_1(\mathcal{I}, f) + L_1(\mathcal{J}, f). \tag{25}$$

Proof. By the definition 6, there exists integer t^* with $0 \le t^* \le u(f)$ such that

$$L_1(\mathcal{I} \cup \mathcal{J}, f) = L_0\Big((\mathcal{I} \cup \mathcal{J})^f_{[0,t^*]}\Big) + L_0\Big((\mathcal{I} \cup \mathcal{J})^f_{[t^*,u(f)]}\Big). \tag{26}$$

Note that $(\mathcal{I} \cup \mathcal{J})^f_{[0,t^*]} = (\mathcal{I})^f_{[0,t^*]} \cup (\mathcal{J})^f_{[0,t^*]}$ and $(\mathcal{I})^f_{[0,t^*]} \cap (\mathcal{J})^f_{[0,t^*]} = \emptyset$, so by Lemma B.1, we have

$$L_0\Big((\mathcal{I} \cup \mathcal{J})^f_{[0,t^*]}\Big) \ge L_0(\mathcal{I}^f_{[0,t^*]}) + L_0(\mathcal{J}^f_{[0,t^*]}). \tag{27}$$

By a similar argument we have

$$L_0\Big((\mathcal{I} \cup \mathcal{J})^f_{[t^*,u(f)]}\Big) \ge L_0(\mathcal{I}^f_{[t^*,u(f)]}) + L_0(\mathcal{J}^f_{[t^*,u(f)]}). \tag{28}$$

By (26), (27) and (28) we have

$$\begin{aligned}
L_1(\mathcal{I} \cup \mathcal{J}, f) \;&\ge\; L_0(\mathcal{I}^f_{[0,t^*]}) + L_0(\mathcal{I}^f_{[t^*,u(f)]}) + L_0(\mathcal{J}^f_{[0,t^*]}) + L_0(\mathcal{J}^f_{[t^*,u(f)]}) \\
&\ge\; \min_{0 \le t \le u(f)} \big\{L_0(\mathcal{I}^f_{[0,t]}) + L_0(\mathcal{I}^f_{[t,u(f)]})\big\} + \min_{0 \le t \le u(f)} \big\{L_0(\mathcal{J}^f_{[0,t]}) + L_0(\mathcal{J}^f_{[t,u(f)]})\big\} \\
&=\; L_1(\mathcal{I}, f) + L_1(\mathcal{J}, f).
\end{aligned}$$

This completes the proof. □

A similar inequality also holds for L_2, as shown below.

Lemma B.3. *For any disjoint sets $\mathcal{I}, \mathcal{J} \subseteq [n]$ and any $\phi \in \Phi$, it holds*

$$L_2(\mathcal{I} \cup \mathcal{J}, \phi) \geq L_2(\mathcal{I}, \phi) + L_2(\mathcal{J}, \phi).$$

Proof. Given $\phi = (f_0, [a, b], f_1, f_2) \in \Phi$, from the equality (7), there exists an integer $t^* \in [a, b]$ such that

$$L_2(\mathcal{I} \cup \mathcal{J}, \phi) = L_1((\mathcal{I} \cup \mathcal{J})^{f_0}_{[0,t^*]}, f_1) + L_1((\mathcal{I} \cup \mathcal{J})^{f_0}_{[t^*,u(f_0)]}, f_2). \tag{29}$$

Note that $(\mathcal{I} \cup \mathcal{J})^{f_0}_{[0,t^*]} = (\mathcal{I})^{f_0}_{[0,t^*]} \cup (\mathcal{J})^{f_0}_{[0,t^*]}$ and $(\mathcal{I})^{f_0}_{[0,t^*]} \cap (\mathcal{J})^{f_0}_{[0,t^*]} = \emptyset$, so by Lemma B.2, we have

$$L_1\Big((\mathcal{I} \cup \mathcal{J})^{f_0}_{[0,t^*]}, f_1\Big) \geq L_1(\mathcal{I}^{f_0}_{[0,t^*]}, f_1) + L_1(\mathcal{J}^{f_0}_{[0,t^*]}, f_1). \tag{30}$$

By a similar argument we have

$$L_1\Big((\mathcal{I} \cup \mathcal{J})^{f_0}_{[t^*,u(f_0)]}, f_2\Big) \geq L_1(\mathcal{I}^{f_0}_{[t^*,u(f_0)]}, f_2) + L_1(\mathcal{J}^{f_0}_{[t^*,u(f_0)]}, f_2). \tag{31}$$

Combining (29), (30) and (31), we have

$$\begin{aligned}
L_2(\mathcal{I} \cup \mathcal{J}, \phi) \;&\geq\; L_1(\mathcal{I}^{f_0}_{[0,t^*]}, f_1) + L_1(\mathcal{I}^{f_0}_{[t^*,u(f_0)]}, f_2) + L_1(\mathcal{J}^{f_0}_{[0,t^*]}, f_1) + L_1(\mathcal{J}^{f_0}_{[t^*,u(f_0)]}, f_2) \\
&\geq\; \min_{0 \leq t \leq u(f_0)} \big\{L_1(\mathcal{I}^{f_0}_{[0,t]}, f_1) + L_1(\mathcal{I}^{f_0}_{[t,u(f_0)]}, f_2)\big\} + \min_{0 \leq t \leq u(f_0)} \big\{L_1(\mathcal{J}^{f_0}_{[0,t]}, f_1) + L_1(\mathcal{J}^{f_0}_{[t,u(f_0)]}, f_2)\big\} \\
&=\; L_2(\mathcal{I}, \phi) + L_2(\mathcal{J}, \phi).
\end{aligned}$$

This completes the proof.

□

B.2. Proof of Lemma 3.1

Proof. By definition (7) we have

$$L_2(\mathcal{I}, f_0, [a, b], f_1, f_2) = \min_{a \leq t \leq b} \big\{L_1(\mathcal{I}^{f_0}_{[0,t]}, f_1) + L_1(\mathcal{I}^{f_0}_{[t,u(f_0)]}, f_2)\big\}.$$

Let t^* be the integer in $[a, b]$ such that

$$L_2(\mathcal{I}, f_0, [a, b], f_1, f_2) = L_1(\mathcal{I}^{f_0}_{[0,t^*]}, f_1) + L_1(\mathcal{I}^{f_0}_{[t^*,u(f_0)]}, f_2). \tag{32}$$

Since there exists $j^* \in [s']$ such that $t_{j^*-1} \leq t^* \leq t_{j^*}$, by Lemma B.2 we have

$$L_1(\mathcal{I}^{f_0}_{[0,t^*]}, f_1) \geq L_1(\mathcal{I}^{f_0}_{[0,t_{j^*-1}]}, f_1), \quad \text{and} \quad L_1(\mathcal{I}^{f_0}_{[t^*,u(f_0)]}, f_2) \geq L_1(\mathcal{I}^{f_0}_{[t_{j^*},u(f_0)]}, f_2). \tag{33}$$

Combining (32) and (33) we have

$$\begin{aligned}
L_2(\mathcal{I}, f_0, [a, b], f_1, f_2) &\geq L_1(\mathcal{I}^{f_0}_{[0,t_{j^*-1}]}, f_1) + L_1(\mathcal{I}^{f_0}_{[t_{j^*},u(f_0)]}, f_2) \\
&\geq \min_{j \in [s']} \big\{L_1(\mathcal{I}^{f_0}_{[0,t_{j-1}]}, f_1) + L_1(\mathcal{I}^{f_0}_{[t_j,u(f_0)]}, f_2)\big\} \\
&= \widehat{L}_2(\mathcal{I}, f_0, [a, b], f_1, f_2, s').
\end{aligned}$$

□

B.3. Proof of Lemma 3.2

Proof. By definition it is trivial that $W_0(\mathcal{I},\phi) \le W_{1,s'}(\mathcal{I},\phi)$ for any $s' \le b-a$; By Lemma 3.1, we know that $W_{1,s'}(\mathcal{I},\phi) \le W_2(\mathcal{I},\phi)$. In the following, we prove that $W_2(\mathcal{I},\phi) \le L_2(\mathcal{I},\phi)$. Suppose $\phi = (f_0,[a,b],f_1,f_2)$ with $f_0,f_1,f_2 \in [p]$ and $0 \le a \le b \le u(f_0)$. Note that

$$\begin{aligned} L_2(\mathcal{I},\phi) &= \min_{t\in[a,b]} \Big\{ L_1(\mathcal{I}^{f_0}_{[0,t]}, f_1) + L_1(\mathcal{I}^{f_0}_{[t,u(f_0)]}, f_2) \Big\} \\ &\ge \min_{t\in[a,b]} \Big\{ L_1(\mathcal{I}^{f_0}_{[0,a]}, f_1) + L_1(\mathcal{I}^{f_0}_{[a,t]}, f_1) + L_1(\mathcal{I}^{f_0}_{[t,b]}, f_2) + L_1(\mathcal{I}^{f_0}_{[b,u(f_0)]}, f_2) \Big\} \\ &= L_1(\mathcal{I}^{f_0}_{[0,a]}, f_1) + L_1(\mathcal{I}^{f_0}_{[b,u(f_0)]}, f_2) + L_2(\mathcal{I}^{f_0}_{[a,b]}, f_0, [a,b], f_1, f_2) \;=\; W_2(\mathcal{I},\phi), \end{aligned}$$

where the inequality follows from Lemma B.2. □

B.4. Proof of Proposition 4.1

Proof. Note that

$$\mathcal{T}_2(f_0,[a,b],F_1,F_2) = \bigcup_{j=1}^{s} \mathcal{T}_2(f_0,[t_{j-1},t_j],F_1,F_2). \tag{34}$$

So we have

$$\mathcal{T}_2(f_0,[a,b],F_1,F_2) \setminus \bigcup_{j=1}^{s} \mathcal{T}_2(f_0,[t_{j-1},t_j],F_{1,j},F_{2,j}) = \bigcup_{j=1}^{s} \Big(\mathcal{T}_2(f_0,[t_{j-1},t_j],F_1,F_2) \setminus \mathcal{T}_2(f_0,[t_{j-1},t_j],F_{1,j},F_{2,j}) \Big).$$

Suppose (for contradiction) that an optimal solution T^* is in the l.h.s. of the above set, then there exists $j \in [s]$ such that

$$T^* \in \mathcal{T}_2(f_0,[t_{j-1},t_j],F_1,F_2) \setminus \mathcal{T}_2(f_0,[t_{j-1},t_j],F_{1,j},F_{2,j}). \tag{35}$$

Then we know $f_O(T^*) = f_0$, $f_L(T^*) \in F_1$, $f_R(T^*) \in F_2$, and at least one of the following two cases hold:

$$(i)\ f_L(T^*) \in F_1 \setminus F_{1,j}; \qquad (ii)\ f_R(T^*) \in F_2 \setminus F_{2,j}. \tag{36}$$

If (i) holds, then we have

$$\begin{aligned} L_2([n],f_0,[t_{j-1},t_j],f_L(T^*),f_R(T^*)) \;&\ge\; W([n],f_0,[t_{j-1},t_j],f_L(T^*),f_R(T^*)) \\ &\ge\; \min_{f_2\in F_2} \big\{ W([n],f_0,[t_{j-1},t_j],f_L(T^*),f_2) \big\} \;>\; U, \end{aligned} \tag{37}$$

where the first inequality is by the assumption that $W(\mathcal{I},\phi) \le L_2(\mathcal{I},\phi)$ for all $\phi \in \Phi$; the last inequality is by the definition of $F_{1,j}$. Note that $L_2([n],f_0,[t_{j-1},t_j],f_L(T^*),f_R(T^*))$ is the optimal value of (2), this is a contradiction to the assumption that U is an upper bound of the optimal value of (2). If (ii) holds, by a similar argument as shown above we have a contradiction.

□

B.5. Proof of Theorem 4.2

Proof. Define

$$C_k := \max_{(f_0,[a,b],F_1,F_2)\in \mathtt{AL}^{(k)}} \{b-a\}$$

as the length of the longest interval over all tuples in $\mathtt{AL}^{(k)}$. As stated in Section 4.2, in each iteration Algorithm 1 splits space $\mathcal{T}_2(f_0,[a,b],F_1,F_2)$ into at most s subsets

$$\bigcup_{j\in[s]} \mathcal{T}_2(f_0,[t_{j-1},t_j],F_{1,j},F_{2,j}).$$

Using the definition of s-equi-spaced points in $[a,b]$, and note that for any real numbers s_1, s_2 it holds $\lfloor s_1 \rfloor - \lfloor s_2 \rfloor \leq \lceil s_1 - s_2 \rceil$, we have

$$t_j - t_{j-1} \leq \lceil a + (j/s)(b-a) - a - ((j-1)/s)(b-a) \rceil = \left\lceil \frac{b-a}{s} \right\rceil \quad \text{for all } j \in [s].$$

Namely, the length of the interval $[a,b]$ in the tuple $(f_0, [a,b], F_1, F_2)$ reduces to $1/s$ of its original value in every iteration. Therefore, $C_k \leq \left\lceil \frac{C_{k-1}}{s} \right\rceil$ for any $k \geq 1$. Applying this inequality recursively, it holds that for any positive integers k_0, m, $C_{k_0} \leq m$ as long as $C_0 \leq m s^{k_0}$. Setting $k_0 = \lceil log_s(n) \rceil - 1$ and $m = \lceil \frac{C_0}{s^{k_0}} \rceil$ yields

$$C_{k_0} \leq \left\lceil \frac{C_0}{s^{k_0}} \right\rceil \leq \left\lceil \frac{n}{s^{k_0}} \right\rceil \leq s.$$

If $\texttt{AL}^{(k_0)}$ is not empty, then any tuple $(f_0, [a,b], F_1, F_2)$ in it satisfies $b - a \leq s$. The algorithm will then perform the exhaustive search method to each tuple in $\texttt{AL}^{(k_0)}$ in the iteration $k_0 + 1$. Hence, $\texttt{AL}^{(k_0+1)} = \emptyset$, and the algorithm terminates in at most $k_0 + 1 = \lceil log_s(n) \rceil$ iterations.

Now, we prove that the algorithm yields an optimal solution. Suppose (by contradiction) that for some k_0, $\texttt{AL}^{(k_0)}$ becomes empty and the algorithm offers an sub-optimal solution. This indicates that in some iteration $k \leq k_0$, (at least one of if there exist multiple optimal solutions) the optimal solution is discarded during Step2, otherwise the algorithm would examine the loss of the optimal solution and record it since it is optimal. This contradicts to Proposition 4.1, which guarantees that no optimal solution will be eliminated in Step2. Therefore, the algorithm will give the optimal solution. The proof is completed.

□

B.6. Proof of Lemma 4.3

Proof. Note that for all these three choices of W, Step 1 is the same.

For the cost of Step 1, note that

$$\begin{aligned} U' &= \min_{f_1 \in F_1, f_2 \in F_2, 0 \leq j \leq s} \left\{ L_1([n]^{f_0}_{[0,t_j]}, f_1) + L_1([n]^{f_0}_{[t_j, u(f_0)]}, f_2) \right\} \\ &= \min_{0 \leq j \leq s} \left\{ \min_{f_1 \in F_1} \{ L_1([n]^{f_0}_{[0,t_j]}, f_1) \} + \min_{f_2 \in F_2} \{ L_1([n]^{f_0}_{[t_j, u(f_0)]}, f_2) \} \right\}. \end{aligned} \tag{38}$$

Recall that $L_1(\mathcal{I}, f)$ can be computed within $\widetilde{O}(|\mathcal{I}|)$ operations, so U' can be computed within $\widetilde{O}(n\tilde{p}s)$ operations.

The major cost of Step 2 lies in the computation of

$$\min_{f_2 \in F_2} W([n], \phi^j_{f_1, f_2}) = \min_{f_2 \in F_2} \left\{ W([n], f_0, [t_{j-1}, t_j], f_1, f_2) \right\} \quad \text{for all } f_1 \in F_1 \text{ and } j \in [s], \tag{39}$$

and

$$\min_{f_1 \in F_1} W([n], \phi^j_{f_1, f_2}) = \min_{f_1 \in F_1} \left\{ W([n], f_0, [t_{j-1}, t_j], f_1, f_2) \right\} \quad \text{for all } f_2 \in F_2 \text{ and } j \in [s]. \tag{40}$$

Once (39) and (40) have been computed, the remaining cost of Step 2 can be bounded by $O(\tilde{p}s)$. Below we show the costs of (39) and (40), under different choices of W.

(1) If $W = W_0$, we have

$$\begin{aligned} \min_{f_2 \in F_2} \left\{ W([n], f_0, [t_{j-1}, t_j], f_1, f_2) \right\} &= \min_{f_2 \in F_2} \left\{ L_1([n]^{f_0}_{[0,t_{j-1}]}, f_1) + L_1([n]^{f_0}_{[t_j, n]}, f_2) \right\} \\ &= L_1([n]^{f_0}_{[0,t_{j-1}]}, f_1) + \min_{f_2 \in F_2} \left\{ L_1([n]^{f_0}_{[t_j, n]}, f_2) \right\}, \end{aligned} \tag{41}$$

where the first equality makes use of the definition of W_0 and the assumption that $u(f_0) = n$. By the expression above, we know that computing (39) requires at most $\widetilde{O}(n\tilde{p}s)$ operations. By a similar argument, computing (40) also requires at most $\widetilde{O}(n\tilde{p}s)$ operations. Hence the cost of Step 2 is bounded by $\widetilde{O}(n\tilde{p}s)$.

(2) If $W = W_{1,s'}$ for some integer $s' \leq b - a$, we have

$$\begin{aligned} &\min_{f_2 \in F_2} \left\{ W([n], f_0, [t_{j-1}, t_j], f_1, f_2) \right\} \\ &= \min_{f_2 \in F_2} \Big\{ L_1([n]^{f_0}_{[0,t_{j-1}]}, f_1) + L_1([n]^{f_0}_{[t_j,n]}, f_2) + \widehat{L}_2([n]^{f_0}_{[t_{j-1},t_j]}, f_0, [t_{j-1}, t_j], f_1, f_2, s') \Big\}, \end{aligned} \tag{42}$$

where we have used the assumption that $u(f_0) = n$. Let $(r_0, ..., r_{s'})$ be the almost s'-equi-spaced integers in $[t_{j-1}, t_j]$, then by the definition of $\widehat{L}_2$ we have

$$\begin{aligned} & \min_{f_2 \in F_2} \left\{ W([n], f_0, [t_{j-1}, t_j], f_1, f_2) \right\} \\ = & \min_{f_2 \in F_2, k \in [s']} \Big\{ L_1([n]^{f_0}_{[0,t_{j-1}]}, f_1) + L_1([n]^{f_0}_{[t_{j-1},r_{k-1}]}, f_1) + L_1([n]^{f_0}_{[r_k,t_j]}, f_2) + L_1([n]^{f_0}_{[t_j,n]}, f_2) \Big\} \\ = & \; L_1([n]^{f_0}_{[0,t_{j-1}]}, f_1) \; + \; \min_{k \in [s']} \Big\{ L_1([n]^{f_0}_{[t_{j-1},r_{k-1}]}, f_1) + \min_{f_2 \in F_2} \left\{ L_1([n]^{f_0}_{[r_k,t_j]}, f_2) + L_1([n]^{f_0}_{[t_j,n]}, f_2) \right\} \Big\} \\ := & \; J_1(j, f_1) \; + \; \min_{k \in [s']} \Big\{ J_3(j, f_1, k) + \min_{f_2 \in F_2} \left\{ J_4(j, f_2, k) + J_2(j, f_2) \right\} \Big\}. \end{aligned} \tag{43}$$

where $J_1(j, f_1) := L_1([n]^{f_0}_{[0,t_{j-1}]}, f_1)$; $J_2(j, f_2) := L_1([n]^{f_0}_{[t_j,n]}, f_2)$; $J_3(j, f_1, k) := L_1([n]^{f_0}_{[t_{j-1},r_{k-1}]}, f_1)$ and $J_4(j, f_2, k) := L_1([n]^{f_0}_{[r_k,t_j]}, f_2)$, and we highlighted the dependence on j, f_1, f_2, k. Note that:

- $\{J_1(j, f_1)\}_{j \in [s], f_1 \in F_1}$ can be computed with $\widetilde{O}(n\tilde{p}s)$ operations.
- $\{J_2(j, f_2)\}_{j \in [s], f_2 \in F_2}$ can be computed with $\widetilde{O}(n\tilde{p}s)$ operations.
- $\{J_3(j, f_1, k)\}_{j \in [s], f_1 \in F_1, k \in [s']}$ can be computed with $\widetilde{O}(\tilde{p}s'(t_j - t_{j-1})s) = \widetilde{O}(\tilde{p}s'(b - a))$ operations.
- $\{J_4(j, f_2, k)\}_{j \in [s], f_2 \in F_2, k \in [s']}$ can be computed with $\widetilde{O}(\tilde{p}s'(t_j - t_{j-1})s) = \widetilde{O}(\tilde{p}s'(b - a))$ operations.

After the values above have been computed and maintained in memory,

- $\Big\{ \min_{f_2 \in F_2} \left\{ J_4(j, f_2, k) + J_2(j, f_2) \right\} \Big\}_{j \in [s], k \in [s']}$ can be computed with $O(\tilde{p}ss')$ operations.

Based on this, we know

- Computing $\min_{k \in [s']} \Big\{ J_3(j, f_1, k) + \min_{f_2 \in F_2} \left\{ J_4(j, f_2, k) + J_2(j, f_2) \right\} \Big\}$ for all $j \in [s]$ and $f_1 \in F_1$ requires at most $O(\tilde{p}ss')$ operations.

With the analysis above, the computation of (39) requires at most $\widetilde{O}(n\tilde{p}s + \tilde{p}s'(b - a))$ operations. By a similar analysis, the computation of (40) also requires at most $\widetilde{O}(n\tilde{p}s + \tilde{p}s'(b - a))$ operations.

(3) If $W = W_2$, the analysis is the same as the analysis for $W = W_{1,s'}$, with s' being of the same order as $(b - a)/s$.

(4) If $W = L_2$, it holds

$$\begin{aligned} \min_{f \in F_2} W([n], f_0, [t_{j-1}, t_j], f_1, f_2) \; &= \; \min_{f \in F_2} L_2([n], f_0, [t_{j-1}, t_j], f_1, f_2) \\ &= \; \min_{f_2 \in F_2, t_{j-1} \leq t \leq t_j} \Big\{ L_1([n]^{f_0}_{[0,t]}, f_1) + L_1([n]^{f_0}_{[t,n]}, f_2) \Big\}. \end{aligned} \tag{44}$$

For a fixed $j \in [s]$, computing the expression above requires $\widetilde{O}((t_j - t_{j-1})\tilde{p}n)$ operations, so the computation of (39) takes $\widetilde{O}(n\tilde{p}(b - a))$ operations. By a similar analysis, the computation of (40) also takes $\widetilde{O}(n\tilde{p}(b - a))$ operations.

□

C. `Quant-BnB` for depth-3 optimal regression trees

In the following we present details on `Quant-BnB` for depth-3 optimal regression trees.

C.1. Notations and preliminaries

Let $\mathcal{T}_3$ be the set of all decision trees with depth 3 whose splitting thresholds are in $\{\tilde{w}_t^f\}_{f\in[p],0\le t\le u(f)}$. The problem of optimal regression tree with depth 3 can be formulated as

$$\min_{T\in\mathcal{T}_3}\sum_{i=1}^{n}\ell(y_i,T(x_i)). \tag{45}$$

We use the notations shown in Figure 3 to denote the nodes in a tree $T\in\mathcal{T}_3$. For $S\in\{O,L,R,LL,LR,RL,RR\}$, let $(f_S(T),t_S(T))$ denote the splitting rule for $N_S(T)$.

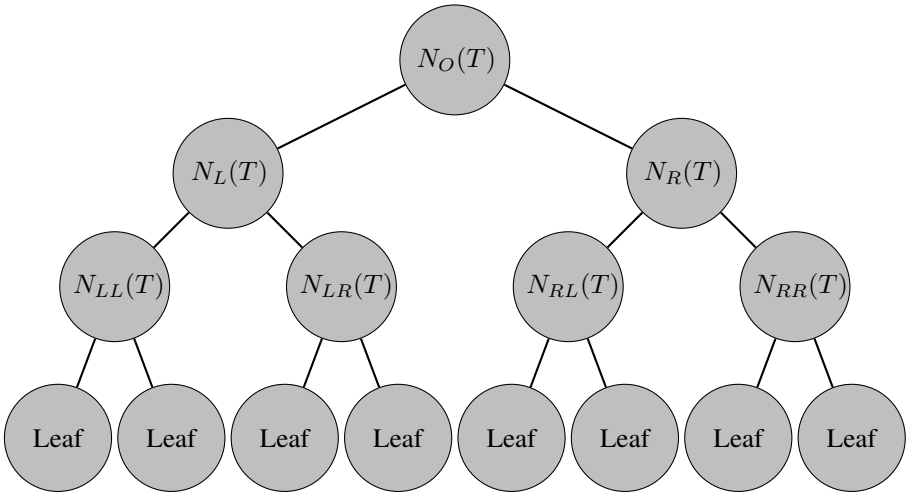


Figure 3. Decision tree with depth 3.

Given $f_0\in[p]$, integers a and b with $0\le a\le b\le u(f_0)$ and $\phi_1,\phi_2\in\Phi$ with $\phi_1=(f_1,[a_1,b_1],f_{1,1},f_{1,2})$ and $\phi_2=(f_2,[a_2,b_2],f_{2,1},f_{2,2})$, define

$$\mathcal{T}_3(f_0,[a_0,b_0],\phi_1,\phi_2) \tag{46}$$

to be the set of all trees $T\in\mathcal{T}_3$ satisfying: $f_O(T)=f_0$, $t_O\in[a_0,b_0]$; $f_L(T)=f_1$, $t_L(T)\in[a_1,b_1]$, $f_{LL}(T)=f_{1,1}$, $f_{LR}(T)=f_{1,2}$; $f_R(T)=f_2$, $t_R(T)\in[a_2,b_2]$, $f_{RL}(T)=f_{2,1}$ and $f_{RR}(T)=f_{2,2}$.

For $\Phi_1,\Phi_2\subseteq\Phi$, define

$$\mathcal{T}_3(f_0,[a_0,b_0],\Phi_1,\Phi_2):=\bigcup_{\phi_1\in\Phi_1,\phi_2\in\Phi_2}\mathcal{T}_3(f_0,[a_0,b_0],\phi_1,\phi_2). \tag{47}$$

For any $\phi=(f_0,[a,b],f_1,f_2)\in\Phi$, and for a given positive integer $s\le b-a$, let $(t_0,...,t_s)$ be almost equi-spaced in $[a,b]$, and define $\phi^{s,j}:=(f_0,[t_{j-1},t_j],f_1,f_2)$ for any $j\in[s]$.

C.2. Quantile-based pruning

In this section, we focus on a subset of trees $\mathcal{T}_3(f_0,[a_0,b_0],\Phi_1,\Phi_2)$, and discuss how to replace it with a smaller search space without missing the optimal solution.

Proposition C.1. *Let W and V be two functions on $2^{[n]}\times\Phi$ satisfying $W(\mathcal{I},\phi)\le L_2(\mathcal{I},\phi)\le V(\mathcal{I},\phi)$ for all $\mathcal{I}\subseteq[n]$ and $\phi\in\Phi$. Let U be an upper bound of the optimal value of (45). Given a subset $\mathcal{T}_3(f_0,[a_0,b_0],\Phi_1,\Phi_2)$ with $f_0\in[p]$, $0\le a_0\le b_0\le u(f_0)$ and $\Phi_1,\Phi_2\subseteq\Phi$; let $(t_0,...,t_s)$ be almost s-equi-spaced in $[a_0,b_0]$. For each $j\in[s]$, define*

$$\begin{aligned}\Phi_{1,j}:=\Big\{\phi_1^{s,j'}\;\Big|\;&\phi_1\in\Phi_1,\ j'\in[s],\ W([n]^{f_0}_{[0,t_{j-1}]},\phi_1^{s,j'})\le\min_{\phi\in\Phi_1}V([n]^{f_0}_{[0,t_j]},\phi),\\ &W([n]^{f_0}_{[0,t_{j-1}]},\phi_1^{s,j'})+\min_{\phi\in\Phi_2}W([n]^{f_0}_{[t_j,u(f_0)]},\phi)\ \le\ U\Big\}\end{aligned} \tag{48}$$

and

$$\Phi_{2,j} := \Big\{\phi_2^{s,j''} \;\Big|\; \phi_2 \in \Phi_2,\ j'' \in [s],\ W([n]^{f_0}_{[t_j,u(f_0)]}, \phi_2^{s,j''}) \le \min_{\phi\in\Phi_2} V([n]^{f_0}_{[t_{j-1},u(f_0)]}, \phi), \\ W([n]^{f_0}_{[t_j,u(f_0)]}, \phi_2^{s,j''}) + \min_{\phi\in\Phi_1} W([n]^{f_0}_{[0,t_{j-1}]}, \phi) \ \le\ U\Big\}. \tag{49}$$

Then any optimal solution of (45) *is not in*

$$\mathcal{T}_3(f_0, [a_0, b_0], \Phi_1, \Phi_2) \setminus \bigcup_{j=1}^{s} \mathcal{T}_3(f_0, [t_{j-1}, t_j], \Phi_{1,j}, \Phi_{2,j}). \tag{50}$$

Proof. Let T^* be any optimal solution of (50). It suffices to prove that: if $T^* \in \mathcal{T}_3(f_0, [a_0, b_0], \Phi_1, \Phi_2)$, then there exists $j \in [s]$ such that

$$T^* \in \mathcal{T}_3(f_0, [t_{j-1}, t_j], \Phi_{1,j}, \Phi_{2,j}). \tag{51}$$

Suppose $T^* \in \mathcal{T}_3(f_0, [a_0, b_0], \Phi_1, \Phi_2)$, then $t_O(T^*) \in [a_0, b_0]$, and there exist integers a_1, b_1, a_2, b_2 such that $t_L(T^*) \in [a_1, b_1]$, $t_R(T^*) \in [a_2, b_2]$, and

$$\phi_1 := (f_L(T^*), [a_1, b_1], f_{LL}(T^*), f_{LR}(T^*)) \in \Phi_1, \\ \phi_2 := (f_R(T^*), [a_2, b_2], f_{RL}(T^*), f_{RR}(T^*)) \in \Phi_2. \tag{52}$$

Since $[a_0, b_0] = \cup_{j=1}^{s}[t_{j-1}, t_j]$, there exists $j \in [s]$ such that $t_O(T^*) \in [t_{j-1}, t_j]$. Let $(\ell_0, ..., \ell_s)$ be almost equi-spaced in $[a_1, b_1]$. $t_L(T^*) \in [a_1, b_1] = \cup_{i=1}^{s}[\ell_{i-1}, \ell_i]$, so there exists $j' \in [s]$ such that $t_L(T^*) \in [\ell_{j'-1}, \ell_{j'}]$. Note that (by definition)

$$\phi_1^{s,j'} = (f_L(T^*), [\ell_{j'-1}, \ell_{j'}], f_{LL}(T^*), f_{LR}(T^*)), \tag{53}$$

we have

$$W([n]^{f_0}_{[0,t_{j-1}]}, \phi_1^{s,j'}) \overset{(i)}{\le} L_2([n]^{f_0}_{[0,t_{j-1}]}, \phi_1^{s,j'}) \overset{(ii)}{\le} L_2([n]^{f_0}_{[0,t_0(T^*)]}, \phi_1^{s,j'}) \\ \overset{(iii)}{=} \min_{\phi\in\Phi_1} L_2([n]^{f_0}_{[0,t_0(T^*)]}, \phi) \overset{(iv)}{\le} \min_{\phi\in\Phi_1} L_2([n]^{f_0}_{[0,t_j]}, \phi) \overset{(v)}{\le} \min_{\phi\in\Phi_1} V([n]^{f_0}_{[0,t_j]}, \phi), \tag{54}$$

where (i) is because $W(\mathcal{I}, \phi) \le L_2(\mathcal{I}, \phi)$ for any $\mathcal{I} \subseteq [n]$ and $\phi \in \Phi$; (ii) is because of $t_O(T^*) \in [t_{j-1}, t_j]$ and Lemma B.3; (iii) is because T^* is the optimal solution of (45), and the left subtree of T^* (rooted at $N_L(T^*)$) is in $\mathcal{T}_2(\phi_1^{s,j'})$; (iv) is because of $t_O(T^*) \in [t_{j-1}, t_j]$ and Lemma B.3; (v) is because $L_2(\mathcal{I}, \phi) \le V(\mathcal{I}, \phi)$ for any $\mathcal{I} \subseteq [n]$ and $\phi \in \Phi$.

On the other hand,

$$W([n]^{f_0}_{[0,t_{j-1}]}, \phi_1^{s,j'}) + \min_{\phi\in\Phi_2} W([n]^{f_0}_{[t_j,u(f_0)]}, \phi) \\ \overset{(i)}{\le} W([n]^{f_0}_{[0,t_{j-1}]}, \phi_1^{s,j'}) + W([n]_{[t_j,u(f_0)]}, \phi_2) \overset{(ii)}{\le} L_2([n]^{f_0}_{[0,t_{j-1}]}, \phi_1^{s,j'}) + L_2([n]^{f_0}_{[t_j,u(f_0)]}, \phi_2) \\ \overset{(iii)}{\le} L_2([n]^{f_0}_{[0,t_0(T^*)]}, \phi_1^{s,j'}) + L_2([n]^{f_0}_{[t_0(T^*),u(f_0)]}, \phi_2) \overset{(iv)}{=} \min_{T\in\mathcal{T}_3} \sum_{i=1}^{n} \ell(y_i, T(x_i)) \overset{(v)}{\le} U, \tag{55}$$

where (i) is because $\phi_2 \in \Phi_2$ (in (52)); (ii) is because $W(\mathcal{I}, \phi) \le L_2(\mathcal{I}, \phi)$ for any $\mathcal{I} \subseteq [n]$ and $\phi \in \Phi$; (iii) is because of $t_0(T^*) \in [t_{j-1}, t_j]$ and Lemma B.3; (iv) is because T^* is the optimal solution of (45), the left subtree of T^* (rooted at $N_L(T^*)$) is in $\mathcal{T}_2(\phi_1^{s,j'})$ and the right subtree of T^* (rooted at $N_R(T^*)$) is in $\mathcal{T}_2(\phi_2)$; (v) is because U is an upper bound of the optimal value of (45).

Combining (54), (55) and note that $\phi_1 \in \Phi_1$ (by (52)) and $j' \in [s]$, we have

$$(f_L(T^*), [\ell_{j'-1}, \ell_{j'}], f_{LL}(T^*), f_{LR}(T^*)) = \phi_1^{s,j'} \in \Phi_{1,j}. \tag{56}$$

Let $(r_0, ..., r_s)$ be almost equi-spaced in $[a_2, b_2]$. Since $t_R(T^*) \in [a_2, b_2] = \cup_{i=1}^{s}[r_{i-1}, r_i]$, so there exists $j'' \in [s]$ such that $t_R(T^*) \in [r_{j''-1}, r_{j''}]$. By a similar argument as the proof of (56), it can be proved that

$$(f_R(T^*), [r_{j''-1}, r_{j''}], f_{RL}(T^*), f_{RR}(T^*)) \in \Phi_{2,j}. \tag{57}$$

By (56) and (57), and recall that $f_O(T^*) = f_0$ and $t_O(T^*) \in [t_{j-1}, t_j]$, we have

$$T^* \in \mathcal{T}_3(f_0, [t_{j-1}, t_j], \Phi_{1,j}, \Phi_{2,j}). \tag{58}$$

This completes the proof of Proposition C.1. □

Note that in the definition of $\Phi_{1,j}$, two inequalities are needed to be satisfied. The first inequality corresponds to a pruning procedure when only considering the left depth-2 subtree rooted at $N_L(T)$ of a tree T; the second inequality corresponds to a pruning procedure considering the whole depth-3 tree. A similar argument holds for the definition of $\Phi_{2,j}$. Note that this is slightly more intricate than the case for fitting depth-2 trees, where only one inequality is imposed (see Proposition 4.1).

C.3. Overall strategy

To solve (45), `Quant-BnB` maintains and updates a set $\texttt{AL3}^{(k)}$ (short for "alive") (over iterations $k = 1, 2, ...$) that contains tuples of the form

$$(f_0, [a_0, b_0], \Phi_1, \Phi_2),$$

where $f_0 \in [p]$, $0 \le a_0 \le b_0 \le u(f_0)$ and $\Phi_1, \Phi_2 \subseteq \Phi$. A tuple $(f_0, [a_0, b_0], \Phi_1, \Phi_2)$ is in $\texttt{AL3}^{(k)}$ if (based on the knowledge at iteration k) the optimal solution of (45) is possible to be in the set $\mathcal{T}_3(f_0, [a_0, b_0], \Phi_1, \Phi_2)$. Initially ($k = 0$), all the trees in $\mathcal{T}_3$ are "alive", so we set

$$\texttt{AL3}^{(0)} = \bigcup_{f_0=1}^{p} \left\{(f_0, [0, u(f_0)], \bar{\Phi}_0, \bar{\Phi}_0)\right\}, \tag{59}$$

where

$$\bar{\Phi}_0 = \{(f, [0, u(f)], f_1, f_2) \mid f, f_1, f_2 \in [p]\}. \tag{60}$$

`Quant-BnB` also maintains and updates the best objective value that it has found so far, denoted by U. Initially, we set U to be the value of any feasible solution of (45). At every iteration $k \ge 1$, to update $\texttt{AL3}^{(k)}$ from $\texttt{AL3}^{(k-1)}$, we first set $\texttt{AL3}^{(k)} = \emptyset$. The algorithm then checks all elements in $\texttt{AL3}^{(k-1)}$. For an element $(f_0, [a_0, b_0], \Phi_1, \Phi_2)$ in $\texttt{AL3}^{(k-1)}$, if $b_0 - a_0$ is less than a fixed integer s, then the number of trees in the space is regarded as small, and the algorithm applies an exhaustive search method to examine all candidate trees in the space $\mathcal{T}_3(f_0, [a_0, b_0], \Phi_1, \Phi_2)$. Otherwise, the algorithm conducts the following steps.

Let $(t_0, ..., t_s)$ be almost s-equi-spaced in $[a_0, b_0]$. Let W and V be two functions on $2^{[n]} \times \Phi$ satisfying $W(\mathcal{I}, \phi) \le L_2(\mathcal{I}, \phi) \le V(\mathcal{I}, \phi)$ for all $\mathcal{I} \subseteq [n]$ and $\phi \in \Phi$.

- (Step 1: Update upper bound) Compute

$$U' = \min_{0 \le j \le s, \phi_1 \in \Phi_1, \phi_2 \in \Phi_2} \left\{ V([n]^{f_0}_{[0,t_j]}, \phi_1) + V([n]^{f_0}_{[t_j, u(f_0)]}, \phi_2) \right\}.$$

  If $U' < U$, update $U \leftarrow U'$, and update the corresponding best tree.

- (Step 2: Compute lower bound and prune) Compute $\Phi_{1,j}$ and $\Phi_{2,j}$ as in (48) and (49), and update:

$$\texttt{AL3}^{(k)} = \texttt{AL3}^{(k)} \bigcup \left(\cup_{j=1}^{s} \{(f_0, [t_{j-1}, t_j], \Phi_{1,j}, \Phi_{2,j})\} \right). \tag{61}$$

Above we have discussed the overall strategy which `Quant-BnB` uses for the computation of optimal regression tree with depth 3. Additional attentions need to be paid for the the implementation of the algorithm and the data structure. For example, to maintain the set Φ_1, it may be better to classify the elements in Φ_1 into groups depending on the first two components of the elements, i.e., for $(f_1, [a_1, b_1], f_{1,1}, f_{1,2})$ and $(f'_1, [a'_1, b'_1], f'_{1,1}, f'_{1,2})$ in Φ_1, they are in the same group if $f_1 = f'_1$ and $[a_1, b_1] = [a'_1, b'_1]$. These groupings can reduce the memory usage and make the computations well-organized. A similar argument holds for Φ_2. It is also important to make use of the tree structure and reduce the computational costs of Step 1 and Step 2.

D. Experiments

D.1. Data pre-processing

We use a collection of 16 classification and 11 regression datasets from UCI Machine Learning Repository. Unless specified in the dataset, 80% of data are randomly selected as the training set, and the rest are used for testing. We remove all features that do not assist prediction, i.e., unique identifiers of samples and timestamps recording when data were collected. Table 5 presents a summary of these datasets.

Name	*Task*	number of samples	number of features	class/dim	Name	*Task*	number of samples	number of features	class/dim
avila	*C*	10430	10	12	skin	*C*	196045	3	2
bank	*C*	1097	4	2	wilt	*C*	4339	5	2
bean	*C*	10888	16	7	carbon	*R*	8576	5	3
bidding	*C*	5056	9	2	casp	*R*	36584	9	1
eeg	*C*	11984	14	2	concrete	*R*	824	8	1
fault	*C*	1552	27	7	energy	*R*	15788	28	1
htru	*C*	14318	8	2	fish	*R*	726	6	1
magic	*C*	15216	10	2	gas	*R*	29386	10	1
occupancy	*C*	8143	5	2	grid	*R*	8000	12	1
page	*C*	4378	10	5	news	*R*	31715	59	1
raisin	*C*	720	7	2	qsar	*R*	436	8	1
rice	*C*	3048	7	2	query1	*R*	8000	3	1
room	*C*	8103	16	4	query2	*R*	159874	4	3
segment	*C*	1848	18	7					

Table 5. A summary of datasets used in experiments. *C* and *R* refer to classification and regression task, respectively. For classification tasks class/dim refers to the number of classes; for regression tasks, class/dim refers to the dimension of the target.

Since most state-of-the-art algorithms can only solve datasets with binary features, we perform the same pre-processing procedure as presented in Lin et al. (2020). For a feature $f \in [p]$, recall that $w_1^f < w_2^f < \cdots < w_{u(f)}^f$ denotes all unique values among $\{x_{i,f}\}_{i=1}^n$. We convert feature f to a set of binary features $\{f_j\}_{j=1}^{u(f)-1}$ defined as

$$x_{i,f_j} = \begin{cases} 0 & \text{if } x_{i,f} < (w_j^f + w_{j+1}^f)/2, \\ 1 & \text{otherwise.} \end{cases} \tag{62}$$

Combining $\{f_j\}_{j=1}^{u(f)-1}$ for each $f \in [p]$ yields a dataset with $\sum_{f\in[p]}(u(f)-1)$ binary features. The conversion is equivalent, namely, an optimal tree on the pre-processed dataset can be converted to an optimal tree on the original dataset and vice versa. In the worst case, the converted dataset has $O(np)$ binary features. It is often computationally challenging to solve the optimal decision tree on such a dataset.

An alternative to equivalent-conversion is approximate conversion, which can greatly reduce the number of binary features. However, approximate conversion weakens the characterization capability of the tree model, which may result in non-negligible decrease in training accuracy. Hence, there is a trade-off between training accuracy and computational cost.

We compare the equivalent-conversion conducted in our numerical experiments with an approximate binarising method adopted in Demirović et al. (2020) (denoted by MDLP). MDLP uses a supervised discretisation algorithm based on the minimum description length principle. We take the training accuracy of CART (Breiman et al., 1984) on original datasets as a benchmark. The size of converted datasets and training accuracy are shown in Table 6. For dataset "avila", "bean","fault" "room", "segment" and "skin", the training accuracy on approximate datasets generated by MDLP is close to the optimal accuracy. However, the accuracy on original datasets outperforms that on approximate datasets by a large margin in rest cases. For several datasets, even the training accuracy of CART is comparable to the training accuracy on approximate datasets. We thus conclude that using equivalent-conversion is indispensable for obtaining high-quality trees.

Name	continuous feature	equivalent conversion	MDLP conversion	depth=2			depth=3		
				Opt	approx	CART	Opt	approx	CART
avila	10	22176	2122	54.2%	54.1%	50.7%	58.5%	58.5%	53.2%
bank	4	4078	26	92.5%	92.3%	90.9%	98.3%	97.3%	93.3%
bean	16	170481	428	66.3%	66.2%	65.7%	87.1%	86.9%	77.7%
bidding	9	10240	44	98.1%	98.1%	98.1%	99.3%	98.1%	98.1%
eeg	14	5239	118	66.5%	65.3%	62.5%	70.8%	68.8%	66.6%
fault	27	16327	244	58.3%	58.3%	54.0%	68.2%	67.3%	55.3%
htru	8	101412	92	97.9%	97.8%	97.7%	98.1%	97.9%	97.9%
magic	10	120435	122	80.5%	80.2%	79.4%	83.1%	81.1%	80.1%
occupancy	5	8339	122	98.9%	98.9%	98.9%	99.4%	99.1%	98.9%
page	10	8175	50	95.4%	95.1%	95.4%	97.1%	96.6%	96.4%
raisin	7	5032	18	87.4%	86.5%	86.8%	89.4%	87.5%	86.9%
rice	7	19982	28	93.3%	93.2%	93.0%	93.8%	93.4%	93.3%
room	16	2879	144	94.6%	94.6%	93.2%	99.2%	99.2%	96.8%
segment	18	13129	145	57.5%	57.4%	43.0%	88.7%	88.1%	57.4%
skin	3	765	108	92.7%	92.7%	90.7%	96.9%	96.8%	96.6%
wilt	5	20329	7	99.1%	98.7%	99.1%	99.6%	98.7%	99.3%

Table 6. Training accuracy (in percentage) of optimal classification trees with depth 2 and 3. The third and forth columns provide the numbers of binary features of datasets processed by equivalent-conversion and MDLP, respectively. Opt denotes the training accuracy of the optimal classification tree on original datasets, approx denotes the training accuracy of the optimal classification tree on datasets binarised using MDLP, and CART denotes the training accuracy of CART on original datasets.

D.2. Optimization algorithms

Details and experimental settings of all comparison algorithms are stated below. Unless specified, implementations of algorithms used in our experiments are obtained from their original authors.

Quant-BnB: Our proposed algorithm is written in Julia programming language (v1.6). The parameter s in Algorithm 1 is set to be 3. In Section 6.2 and Section 6.3, we choose $W_{1,s'}$ defined in (13) as the lower bound required in Proposition 4.1. The parameter s' is dynamically selected as $\lfloor \frac{0.6ns}{b-a} \rfloor$ for tuple $\phi = (f_0, [a, b], f_1, f_2)$.

CART (Breiman et al., 1984): We utilize the implementation from Python package scikit-learn.

TAO (Carreira-Perpinán & Tavallali, 2018): We implement TAO in Julia 1.6. TAO uses the solution generated by CART as a warm start.

OCT and **ORT** (Bertsimas & Dunn, 2019) : Since the original code is not available, we implement both methods in Python and call Gurobi9 to solve MIP models. Both methods takes the solution generated by CART as a warm start.

BinOCT (Aglin et al., 2020): BinOCT is written in Python. We slightly modify the code so that the MIP model is solved by Gurobi9. BinOCT uses the solution generated by CART as a warm start.

DL8.5 (Aglin et al., 2020): DL8.5 is written in C++ and is run as an extension of Python.

MurTree (Demirović et al., 2020): MurTree is written in C++ and run as an executable.

FlowOCT and **BenderOCT** (Aghaei et al., 2021): Both methods are implemented in Python. MIP models are solved by Gurobi9.

GOSDT (Lin et al., 2020): GOSDT is written in C++ and run as an executable. GOSDT does not force hard constraints on depth, but instead applies a sparsity coefficient α to control the complexity. As α decrease, GOSDT takes longer time to solve an optimal tree. To facilitate a fair comparison with our algorithm on learning optimal depth-2 (or 3) tree, we test GOSDT with

$$\alpha \in \{0.1, 0.05, 0.02, 0.01, 0.005, 0.002, 0.001\}, \tag{63}$$

and select the smallest α with which GOSDT can learn an optimal tree with depth not greater than 2 (or 3).

Other works for learning optimal trees (e.g., Aghaei et al. (2019)) that do not show noticeable speed advantages are not

mentioned above. We do not consider the comparison with these algorithms as we focus on the efficiency of solving optimal trees,

In addition to BinOCT, MurTree and DL8.5, we also run OCT, FlowOCT, BenderOCT and GOSDT on collected classification datasets. For FlowOCT, BenderOCT and GOSDT, we convert original datasets to binary ones using equivalent-conversion described in Section D.1. None of these methods is able to learn an optimal tree on any of 16 classification datasets, so the results are not displayed.

D.3. Results on regression tasks

We also compare our algorithm with ORT (Bertsimas & Dunn, 2019) on 11 regression datasets. To our best knowledge, ORT is the one of the most effective framework reported in the literature for solving optimal decision trees on regression tasks. The results are displayed in Table 7. `Quant-BnB` successfully solves trees of depth 2 in less than 10 seconds on datasets with thousands of instances, and computes trees of depth 3 in a reasonable time for most cases. In contrast, ORT cannot optimally solve any example in 4 hours. The experiment again confirms the advantage of `Quant-BnB` for solving shallow decision trees on relatively large-scale datasets. The scalability and versatility of our proposed method contribute to the wide applications of optimal decision trees.

Name	(n,p)	depth=2		depth=3	
		Quant-BnB	ORT	Quant-BnB	ORT
carbon	(8576,5)	**0.7**	(20%)	**729**	(423%)
casp	(36584,9)	**4.2**	(14%)	**7609**	(12%)
concrete	(824,8)	**<0.1**	(1.2%)	**125**	(33%)
energy	(15788,28)	**14**	(9.6%)	-	(7.6%)
fish	(726,6)	**<0.1**	(4.5%)	**34**	(36%)
gas	(29386,10)	**1.5**	(1358%)	**421**	(510%)
grid	(8000,12)	**1.2**	(9.3%)	**1293**	(31%)
news	(31715,59)	**349**	(22%)	-	(29%)
qsar	(436,8)	**<0.1**	(5.1%)	**30**	(32%)
query1	(8000,3)	**<0.1**	(38%)	**34**	(73%)
query2	(159874,4)	**9.8**	(97%)	**2896**	OoM

Table 7. Comparison of `Quant-BnB` against ORT. For each dataset, the number of observations and the number of features are provided. Each entry denotes running time in seconds. - refers to time out (4h), OoM refers to out of memory (25GB). Since ORT times out in all cases, we display the relative differences $(L_O - L_Q)/L_Q$ as a percentage instead, where L_O and L_Q are the training errors of ORT and `Quant-BnB`, respectively.